VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2011 Tomáš Režnar

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Demonstrace křivek a ploch v počítačové grafice
Curves and Surfaces in Computer Graphics

2011 Tomáš Režnar

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Režnar**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Demonstrace křivek a ploch v počítačové grafice
Curves and Surfaces in Computer Graphics

Zásady pro vypracování:

Cílem práce je implementovat prostředí, které umožní demonstrovat základní typy křivek a ploch používané v počítačové grafice. Cílem bude zaměřit se na vhodné prostředí a ovládání tak, aby práci bylo možné využít při výuce.
1. Nastudujte problematiku křivek a ploch v počítačové grafice.
2. Vytvořte aplikaci demonstrující vybrané křivky a plochy, včetně jejich vlastností.
3. Práci průběžně konzultujte s vedoucím tak, aby mohla být v budoucnu použita ve výuce.
4. Práci pečlivě teoreticky zdokumentujte, aby v ní bylo možné dále pokračovat.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Martin Němec, Ph.D.**

Datum zadání: 19.11.2010
Datum odevzdání: 06.05.2011

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA OSTRAVA Fakulta elektrotechniky a informatiky

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 6.5.2011

Tomáš Režnar

Poděkování

Děkuji vedoucímu mé bakalářské práce panu Ing. Martinu Němci, Ph.D. za trpělivost, ochotu a cenné rady.

Abstrakt

Cílem této bakalářské práce bylo zaměřit se na návrh a implementaci počítačové aplikace, umožňující demonstrovat vlastnosti vybraných křivek a ploch. Křivky a plochy lze zobrazovat v perspektivní i ortogonální projekci v trojrozměrném prostoru včetně možnosti nastavit polohu kamery. Aplikace umožňuje uživatelům vytvářet vlastní projekty, které je možné načítat a ukládat do souborů. Výsledná aplikace by měla primárně sloužit k výukovým účelům, je navržená tak, aby ji bylo dále možné jednoduše upravovat a rozšiřovat o další křivky a plochy. Součástí je popis vlastností vybraných křivek a ploch a implementace algoritmů pro jejich výpočet.

Klíčová slova

křivka, plocha, počítačová grafika, OpenGL, JOGL

Abstract

This bachelor thesis is focused on designing and implementing computer application, which illustrates attributes of selected computer curves and surfaces. It enables to display curves and surfaces in three-dimensional space with perspective or orthogonal projection with capability of changing camera position. Computer application is able to create projects, which can be loaded from and saved to files. Application is intended for educational purposes. Work is designed in such a way to be easily modified and extended by other curves and surfaces. Part of my work is description of attributes of selected curves and surfaces and implementation of alogirithms for their calculation.

Keywords

curve, surface, computer graphics, OpenGL, JOGL

Seznam použitých zkratek a symbolů

2D	Dvojrozměrný prostor
3D	Třírozměrný prostor
API	Application Programming Interface – popis funkcí a tříd nějaké knihovny, které může programátor využívat
CAD	Computer-aided design – používání počítačových grafických programů pro projektování
CAPI	Curve API – Java knihovna pro výpočet křivek
GPL	General Public License – licence svobodného softwaru
GUI	Graphical User Interface – grafické uživatelské rozhraní
JGEOM	Geometry library – Java knihovna
JOGL	Java oriented graphic layer – Java knihovna tvořící rozhraní k OpenGL
JVM	Java Virtual Machine – sada počítačových programů ke spuštění dalších programů a skriptů.
LGPL	Lesser General Public License
MVC	Model-view-controller – návrhový vzor
NURBS	Non-uniform rational basis spline – matematický model
OpenGL	Open Graphics Library – API pro tvorbu počítačové grafiky
SSJ	Stochastic Simulation in Java – Java knihovna

Obsah

Seznam obrázků

1 Úvod

Počítačová grafika je obor informatiky využívaný v mnoha odvětvích. Křivky a plochy se využívají například v CAD systémech ve stavebnictví, architektuře, elektrotechnice, strojírenství a v mnoha dalších oborech. Ve všech těchto oborech se využívají pro vytvoření grafického díla různé typy křivek a ploch, tyto křivky a plochy se liší svými vlastnostmi. Pro studenty je důležité, aby si kromě vzorců a definic vyzkoušeli, jak daná křivka nebo plocha vypadá a jak její vlastnosti ovlivňují výsledný tvar a chování.

Cílem práce je návrh a implementace počítačové aplikace, která bude primárně používána ve výuce k demonstraci vlastností křivek a ploch využívaných v počítačové grafice. Aby si, například cvičící, mohl předem připravit ukázkové příklady pro účely výuky, bude možné vytvořené křivky a plochy v aplikaci ukládat v podobě projektů. Důležitým prvkem aplikace je její rozšiřitelnost o další křivky a plochy.

Typů křivek a ploch je velké množství, jejich vysvětlení a popis by bylo mimo rozsah této práce, proto je součástí práce výběr vhodných a často používaných křivek a ploch. Pro vytvoření aplikace je nutné vybrat vhodný programovací jazyk a knihovny umožňující výpočet nebo vykreslení křivek a ploch.

Analýze a návrhu aplikace je věnována samostatná kapitola, zde jsou popsány balíčky a třídy aplikace včetně diagramů tříd, jejich fungování a návaznosti. Největší důraz je kladen na popis a návrh tříd starajících se o vykreslování křivek a ploch. V práci je také popsána implementace zajímavých částí aplikace: operace s vektory a body, matematické úlohy související s výpočty a vykreslením.

Součástí je také návod, jak do aplikace přidat další křivky a plochy. V návodu jsou popsány abstraktní třídy křivek a ploch, které sdružují společné vlastnosti. Jsou popsány způsoby rozšíření aplikace děděním z abstraktních tříd. V příloze je uvedeno několik ukázkových příkladů.

1.1 Aktuální stav

V dnešní době sice existují na webu ukázky a příklady různých křivek a ploch, obvykle se ale jedná pouze o omezené použití [1], kde si uživatel nemůže danou křivku a plochu vyzkoušet např. ve 3D prostoru apod., nebo se jedná pouze o příklad obvykle jedné křivky nebo plochy [2] [3], bez možnosti srovnání apod. Proto jsem se pokusil vytvořit aplikaci, která by toto umožňovala a byla vhodným doplňkem výuky počítačové grafiky

2 Vybrané křivky a plochy

Jednou z důležitých částí práce je výběr implementovaných křivek a ploch tak, aby aplikace obsahovala ideálně nejpoužívanější křivky, které se při výuce počítačové grafiky probírají. Samozřejmě nelze popsat všechny existující křivky, avšak součástí návrhu aplikace je možnost ji libovolně rozšiřovat o další typy křivek a ploch.

Mezi vybrané a implementované křivky a plochy patří: Bézierova křivka a plocha, Coonsova kubika, Fergusonova kubika, Hermitova interpolace, Interpolace polynomem, Aproximace polynomem, NURBS křivka a plocha a Coonsova plocha.

2.1 Základní vlastnosti křivek a ploch

Jednotlivé křivky a plochy se mohou lišit svým zadáním, popisem, výpočtem apod. To všechno má samozřejmě vliv na výsledné chování jednotlivých křivek a ploch, v praxi obvykle víme, jaké je zadání a jak by se daná křivka nebo plocha měla chovat, z toho jsme schopni vybrat vhodnou křivku nebo plochu. Je nutné však znát nejen vzorec zadané křivky nebo plochy, ale i její vlastnosti a chování.

2.1.1 Vlastnosti křivek

Křivka je geometrický objekt, který lze vyjádřit třemi způsoby: implicitním, explicitním a parametrickým. Implicitní vyjádření má tvar $F(x, y)=0$, explicitní vyjádření může mít tvar $y=f(x)$ a parametrické vyjádření $Q(t)=[x(t), y(t)]$. Nevýhodou explicitního zadání je, že křivka nemůže nemůže procházet stejným bodem vícekrát.

Křivky lze rozdělit na interpolační a aproximační. Interpolační křivky procházejí svými řídícími body, aproximační křivky nemusí procházet svými řídícími body. Dalším rozdělením křivek může být na rovinné a prostorové. Rovinná křivka je zobrazení $x=x(t),\ y=y(t);\ t\in\langle a, b\rangle$ a prostorová křivka je zobrazení $x=x(t),\ y=y(t),\ z=z(t);\ t\in\langle a, b\rangle$. Další rozdělení křivek by mohlo být podle užití, například ve statistice, technické křivky, v umění, k modelování apod.

Základní vlastností křivky je její spojitost. Rozlišujeme spojitost parametrickou a geometrickou. Parametrická spojitost je označovaná C^n. Křivka daná rovnicí $Q(t)$ je spojitá C^n, pokud má ve všech bodech spojité derivace podle parametru t do řádu n. Geometrická spojitost je označovaná G^n. Křivka je G^0 spojitá, pokud pro všechny segmenty křivky platí, že poslední bod segmentu je totožný s počátečním bodem následujícího segmentu. Křivka je G^1 spojitá, pokud pro všechny segmenty křivky platí, že tečný vektor segmentu je souhlasně kolineární s tečným vektorem segmentu následujícího [4].

2.1.2 Vlastnosti ploch

Stejně jako křivky, lze i plochy vyjádřit implicitním, explicitním a parametrickým způsobem. Implicitní vyjádření má tvar *F(x, y, z)=0*, explicitní vyjádření může mít tvar *z=f(x, y)* a parametrické vyjádření *Q(t)=[x(u, v), y(u, v), z(u, v)]*, kde *u* a *v* jsou parametry plochy.

Plochy rovněž můžeme rozdělit na interpolační a aproximační. Interpolační plochy prochází svými řídícími body, aproximační nemusí. Dalším rozdělením ploch je dělení podle způsobu zadání plochy na plochy zadané sítí bodů a plochy zadané okrajovými křivkami.

Další vlastnosti ploch jsou neměnnost plochy vůči lineárním transformacím a projekcím, ovlivnění plochy změnou jednoho řídícího bodu, zdali plocha prochází krajními řídícími body a jestli plocha leží v konvexní obálce řídících bodů.

Další teorii vlastností křivek a ploch můžete nalézt v [4].

2.2 Vybrané křivky a plochy

Kapitola se zabývá stručným teoretickým popisem vybraných křivek a ploch, který je nutný pro pochopení a implementaci.

2.2.1 Interpolace algebraickým polynomem

Mezi základní křivky určené pro interpolaci patří zajisté interpolace algebraickým polynomem. Algebraický polynom je výraz ve tvaru [5]:

$$p(x)=\sum_{i=0}^{n} a_i x^i = a_0+a_1 x+a_2 x^2+\cdots+a_n x^n \tag{2.1}$$

Křivka zadaná *n + 1* body bude stupně nejvýše *n*, pro každý zadaný bod P_i musí platit, že souřadnice x_i řídícího bodu P_i nesmí být rovna žádné jiné souřadnici *x* všech ostatních řídících bodů. Dosazením řídících bodů do vztahu 2.2 vytvoříme soustavu *n+1* rovnic o *n+1* neznámých, ze které vypočítáme koeficienty polynomu a_0 až a_n:

$$\begin{bmatrix} p(x_0) \\ p(x_1) \\ \vdots \\ p(x_n) \end{bmatrix} = \begin{bmatrix} 1 & x_0^1 & x_0^2 & \cdots & x_0^n \\ 1 & x_1^1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n^1 & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \tag{2.2}$$

Obrázek 2.1: Interpolace polynomem čtvrtého stupně

2.2.2 Hermitova interpolace

Hermitova interpolace je metoda pro výpočet Hermitova polynomu. Křivka je zadaná *n* řídicími body a *m* derivacemi v každém řídícím bodě. Hermitův polynom má stupeň nejvýše $n\cdot(m+1)-1$ [6]. Pokud bude křivka zadaná množinou bodů a jejich prvních derivací, bude stupeň křivky nejvýše $n\cdot(1+1)-1=2n-1$. Vztah pro výpočet Hermitova polynomu zadaného množinou bodů a první derivací v každém bodě:

$$H(x)=a_1x^{n-1}+a_2x^{n-2}+\cdots+a_nx^0$$

$$\begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \\ f'(x_1) \\ f'(x_2) \\ \vdots \\ f'(x_n) \end{bmatrix} = \begin{bmatrix} x_1^{n-1} & x_1^{n-2} & \cdots & x_1^0 \\ ax_2^{n-1} & x_2^{n-2} & \cdots & x_2^0 \\ \vdots & \vdots & & \vdots \\ x_n^{n-1} & x_n^{n-2} & \cdots & x_n^0 \\ (n-1)x_1^{n-2} & (n-2)x_1^{n-3} & \cdots & 0 \\ (n-1)x_2^{n-2} & (n-2)x_2^{n-3} & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ (n-1)x_n^{n-2} & (n-2)x_n^{n-3} & \cdots & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ a_n \end{bmatrix} \tag{2.3}$$

Obrázek 2.2: Hermitova kubika

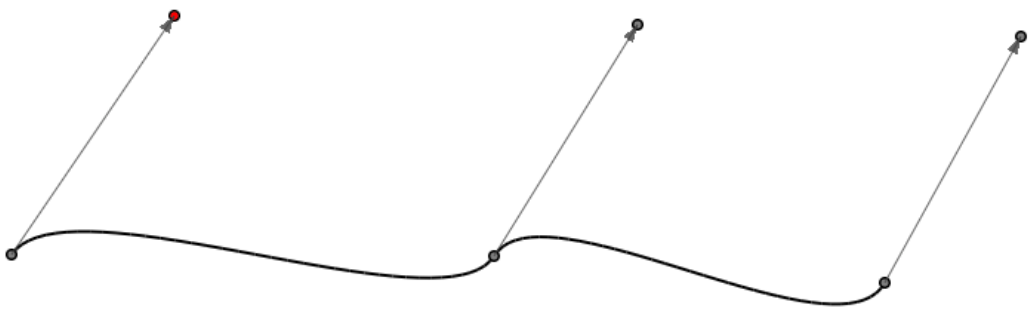
2.2.3 Fergusonova křivka

Fergusonova křivka v některých zdrojích označovaná jako Hermitovská kubika [4]. Fergusonova křivka je dána dvěma krajními řídícími body P_0 a P_1 a dvěma tečnými vektory $\vec{p}_0$ a $\vec{p}_1$. Tečné vektory zadáme pomocí dalších dvou řídících bodů P_2 a P_3.

$$\begin{aligned}\vec{p}_0 &= P_2 - P_0 \\ \vec{p}_1 &= P_3 - P_1\end{aligned} \tag{2.4}$$

Vztah pro výpočet Fergusonovy kubiky:

$$Q(t) = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ \vec{p}_0 \\ \vec{p}_1 \end{bmatrix}; t \in \langle 0,1 \rangle \tag{2.5}$$



Obrázek 2.3: Fergusonovy kubiky napojené se spojitostí C^1

Při navazování křivek docílíme spojitost G^0 pokud druhý řídící bod první křivky je roven prvnímu řídícímu bodu křivky následující. Spojitost G^1 docílíme pokud druhý tečný vektor první křivky a první tečný vektor následující křivky jsou lineárně závislé a zároveň je splněna podmínka pro G^0. Spojitost C^1 docílíme pokud je druhý tečný vektor první křivky je roven prvnímu tečnému vektoru následující křivky a zároveň je splněna podmínka pro G^0.

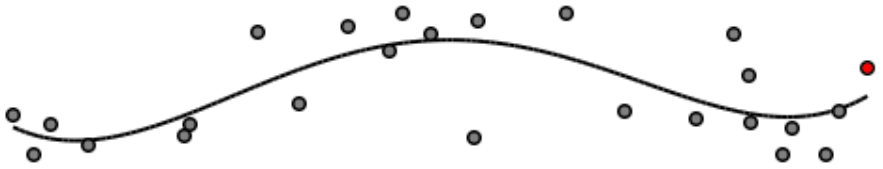
2.2.4 Aproximace polynomem metodou nejmenších čtverců

Aproximační křivka zadaná *n* řídícími body může mít nejvýše stupeň $k < n$ (pro stupeň $k = n - 1$ je průběh křivky shodný s průběhem interpolační křivky algebraickým polynomem zadané shodnými řídícími body). Vztah pro výpočet křivky [7]:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_k x^k \tag{2.6}$$

Koeficienty polynomu vypočteme soustavou rovnic [7]:

$$\begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \vdots \\ \sum_{i=1}^{n} x_i^k y_i \end{bmatrix} = \begin{bmatrix} n & \sum_{i=1}^{n} x_i & \cdots & \sum_{i=1}^{n} x_i^k \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \cdots & \sum_{i=1}^{n} x_i^{k+1} \\ \vdots & \vdots & & \vdots \\ \sum_{i=1}^{n} x_i^k & \sum_{i=1}^{n} x_i^{k+1} & \cdots & \sum_{i=1}^{n} x_i^{2k} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} \tag{2.7}$$



Obrázek 2.4: Aproximace polynomem čtvrtého stupně

2.2.5 Bézierova křivka

Bézierovy křivky a plochy poprvé popsal francouzský inženýr Pierre Bézier, který je využíval k návrhu karosérií automobilů. Dají se zařadit mezi křivky aproximační, jejich hlavní výhodou bylo vhodné chování při modelování. Využívají se například pro definování tvarů fontů písmen v počítači, k definování křivek v CAD a různých grafických aplikacích (například Inkscape, Paint Shop).

Křivka je určena řídicím polygonem, daným *n + 1* řídícími body, kde *n* je stupeň křivky. Vztah pro výpočet Bézierovy křivky [4]:

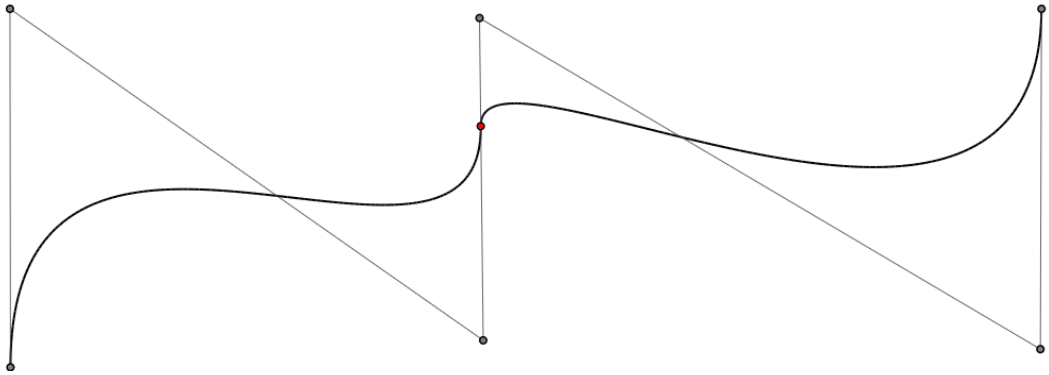
$$Q(t) = \sum_{i=0}^{n} P_i B_i^n(t); t \in \langle 0,1 \rangle \tag{2.8}$$

$B_i^n(t)$ je i-tý Bernsteinův polynom n-tého stupně pro parametr *t*.

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}; kde\ 0^0 = 1\ a\ \binom{n}{0} = 1 \tag{2.9}$$

Bézierova křivka prochází prvním a posledním bodem řídícího polynomu. Je-li křivka prvního stupně, pak je výsledná křivka totožná s obálkou řídicího polygonu. Změna libovolného bodu řídícího polygonu ovlivní tvar celé křivky.

Pro zaručení spojitosti G^0 při spojování více Bézierových křivek musí být poslední bod řídícího polygonu první křivky shodný s prvním bodem řídícího bodu následující křivky.



Obrázek 2.5: Dvě Bézierovy křivky třetího stupně napojené se spojitostí G^1

Pro zaručení spojitosti G^1 musí být splněna podmínka pro G^0 a zároveň musí ležet poslední dva body řídícího polygonu první křivky a první dva body řídícího polygonu následující křivky na společné přímce a zároveň se nebudou tyto body rovnat a budou ležet na přímce v pořadí.

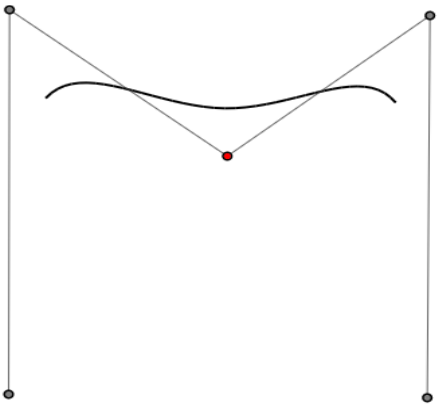
Pro zaručení spojitosti C^1 musí být splněny podmínky pro spojitost G^1 a zároveň musí poslední bod řídícího polygonu první křivky a první bod řídícího polygonu následující křivky ležet uprostřed úsečky definované předposledním bodem řídícího polygonu první křivky a druhým bodem řídícího polygonu následující křivky.

2.2.6 Coonsova B-spline křivka

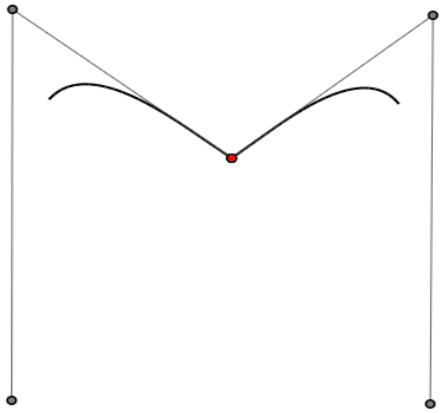
Coonsova křivka je křivka aproximační. Coonsova B-spline křivka se skládá z jednoho a více Coonsových oblouků. Každý oblouk je určen řídícím polygonem se čtyřmi řídicími body. Pro všechny oblouky křivky platí, že poslední tři body řídícího polygonu oblouku jsou shodné s prvními třemi řídícími body následujícího oblouku křivky. Coonsova křivka má spojitost C^2.

Coonsův B-spline neprochází řídícími body, pokud jsou si tři řídící body za sebou rovny (trojnásobný bod), pak jimi křivka prochází a v daném bodě je křivka G^0 spojitá. Coonsův oblouk je dán vztahem [4]:

$$Q(t)=\frac{1}{6}[t^3 \quad t^2 \quad t \quad 1]\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}\begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}; t\in\langle 0,1\rangle \tag{2.10}$$



Obrázek 2.6: Coonsův B-spline



Obrázek 2.7: Coonsův B-spline s trojnásobným bodem

2.2.7 NURBS křivka

NURBS (non-uniform rational basis spline) křivky jsou dané $n + 1$ řídícími body, každý řídící bod má svoji váhu w_i (váha je kladné reálné číslo, čím vyšší hodnota, tím se křivka více přiblíží danému bodu, základní hodnota váhy je $w_i = 1$), řádem křivky k a uzlovým vektorem $U = \{t_0, t_1, \ldots, t_{n+1+k}\}$ [4]. Uzlový vektor musí splňovat podmínky:

- je neklesající posloupností reálných čísel
- nesmí obsahovat více než k shodných prvních nebo posledních složek vektoru
- nesmí obsahovat více než $k - 1$ shodných složek vektoru za sebou, vyjma první a poslední složky.

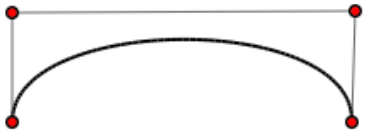
Vztah pro výpočet NURBS křivky:

$$Q(t) = \frac{\sum_{i=0}^{n} w_i P_i N_{i,k}(t)}{\sum_{i=0}^{n} w_i N_{i,k}(t)}; t \in \langle t_k, t_{n+1} \rangle \tag{2.11}$$

Kde P_i jsou řídící body, w_i je váha řídícího bodu P_i, $N_{i,k}(t)$ jsou rekurentní bázové B-spline funkce:

$$\begin{aligned} N_{i,1}(t) &= \begin{cases} 1 & pro\ t_i \le t < t_{i+1} \\ 0 & jinak \end{cases} \\ N_{i,k}(t) &= \frac{t - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t); \end{aligned} \tag{2.12}$$

Ve vztahu platí $\frac{0}{0} = 0$ [8].



Obrázek 2.8: NURBS křivka třetího stupně s neuniformním uzlovým vektorem

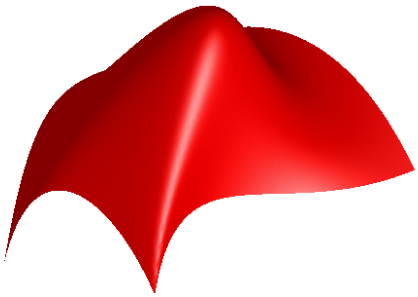
Obrázek 2.9: NURBS křivka třetího stupně s uniformním uzlovým vektorem

2.2.8 Bézierova plocha

Bézierova plocha stupně *n* x *m* je dána *(n + 1)* x *(m + 1)* řídícími body $P_{i,j}$ a vztahem [4]:

$$Q(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m} P_{i,j}\,B_i^n(u)\,B_j^m(v);\; t,u\in\langle 0,1\rangle \tag{2.13}$$

B_i^n a B_i^m jsou Bernsteinovy polynomy ze vztahu 2.9. Pro zjednodušení zadávání plochy v aplikaci bude každá plocha mít stupeň $m = n$.



Obrázek 2.10: Bézierova plocha třetího stupně

2.2.9 Coonsova plocha

Coonsovy plochy patří mezi B-spline plochy. Coonsovy plochy jsou třetího stupně. Jsou dány *4* x *4* řídícími body P_{00} až P_{33}. Jejich spojitost je vždy C^2. Vztah pro výpočet Coonsovy plochy [4]:

$$Q(u,v)=\frac{1}{36}\begin{bmatrix}u^3 & u^2 & u & 1\end{bmatrix}\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}\begin{bmatrix}P_{00} & P_{01} & P_{02} & P_{03}\\ P_{10} & P_{11} & P_{12} & P_{13}\\ P_{20} & P_{21} & P_{22} & P_{23}\\ P_{30} & P_{31} & P_{32} & P_{33}\end{bmatrix}\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}^T\begin{bmatrix}t^3\\ t^2\\ t\\ 1\end{bmatrix};u,v\in\langle 0,1\rangle \tag{2.14}$$

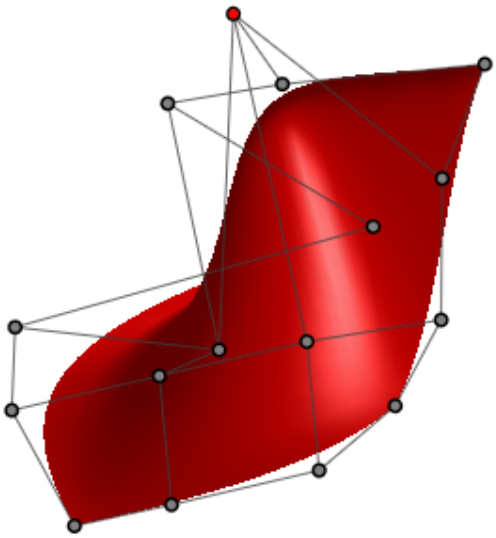
2.2.10 NURBS plocha

NURBS (non-uniform rational basis spline) plochy jsou průmyslový standard pro representaci a návrh geometrických útvarů [9]. NURBS byly vyvinuty kvůli potřebě reprezentovat volné tvary v průmyslu: tvary trupů lodí, povrchy letadel, karoserie automobilů. Nyní jsou běžně používané v počítačových grafických programech [10].

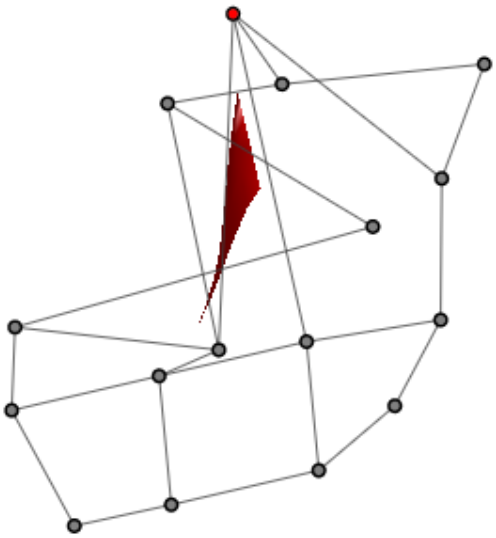
NURBS plocha je určena sítí řídících bodů P ve směru u o počtu $(n + 1)$ a ve směru v o počtu $(m + 1)$, každý řídící bod $P_{i,j}$ má váhu $w_{i,j}$. NURBS plocha má určen řád p ve směru u a řád q ve směru v. Řád plochy musí být menší nebo roven počtu bodů v daném směru: $p \leq (n + 1)$ a $q \leq (m + 1)$. Je dána uzlovými vektory U délky $n + p + 1$ a V délky $m + q + 1$. Uzlové vektory musí splňovat stejné podmínky jako uzlový vektor NURBS křivky uvedený v kapitole 2.2.7. Vztah pro výpočet NURBS plochy:

$$Q(u,v)=\frac{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{i,j} P_{i,j} N_{i,p}(u) N_{j,q}(v)}{\sum_{i=0}^{n}\sum_{j=0}^{m} w_{i,j} N_{i,p}(u) N_{j,q}(v)}; u \in \langle u_p, u_{n+1} \rangle, v \in \langle v_q, v_{m+1} \rangle \tag{2.15}$$

Kde $N_{i,p}$ a $N_{j,q}$ jsou B-spline bázové funkce ze vztahu 2.12.



Obrázek 2.11: NURBS plocha třetího stupně s neuniformními uzlovými vektory U=V=[0,0,0,0,1,1,1,1]



Obrázek 2.12: NURBS plocha třetího stupně s uniformními uzlovými vektory U=V=[0,1,2,3,4,5,6,7]

3 Specifikace požadavků na aplikaci

Tyto požadavky jsou souhrnem požadavků, které vyplývají ze zadání bakalářské práce a z konzultací s vedoucím práce.

- Aplikace má být multiplatformní.
- Aplikace bude sloužit jako pomůcka ve výuce předmětu počítačová grafika, proto výběr implementovaných křivek a ploch by měl být proveden vhodně, tak aby doplňoval teorii křivek a ploch.
- Křivky a plochy by měly jít zobrazit v trojrozměrném prostoru a zobrazení by mělo jít nastavit.
- Aplikace by měla být rozšiřitelná o další typy křivek a ploch.
- Práce s křivkami a plochami formou projektů. Možnost mít otevřené v jeden okamžik více projektů.
- Ukládání a načítání projektů do souborů.
- Jeden projekt se může skládat z více křivek a ploch, kterým je možné nastavovat viditelnost.
- Export aktuálního projektu do obrázku.
- Přepínání zobrazení mezi jedním pohledem a kombinací čtyř pohledů.

4 Použité technologie

Aplikace je naprogramována v jazyku Java ve vývojovém prostředí NetBeans. Java je objektově orientovaný jazyk vyvinutý firmou Sun Microsystems v roce 1995. Je jedním z nejpoužívanějších jazyků na světě [11]. Je implementován velkou řadou zařízení, jako jsou osobní počítače (JavaSE, JavaEE), mobilní telefony (JavaME) a různé chytré zařízení (JavaFX), např. televize, set-top boxy apod.

4.1 Dostupné knihovny

Protože výpočty a vykreslení křivek a ploch jsou výpočetně náročné úlohy (především výpočty ploch), proto je nutné vybrat vhodné knihovny. Kapitola popisuje dostupné knihovny pro jazyk Java, které souvisí s matematickými výpočty nebo vykreslováním křivek a ploch, z těchto knihoven budou vybrány ty knihovny, které jsou svým zaměřením vhodné pro využití v aplikaci.

4.1.1 Curve API (CAPI)

Implementace různých matematických křivek definovaných řídícími body. Podporované křivky jsou Beziérova, B-Spline, obecný Spline, Catmull-Rom Spline, Lagrande, Kubický Spline a NURBS. [12]

Knihovna umožňuje vypočítat a vykreslit pomocí Canvasu širokou škálu křivek, avšak její nevýhodou je chybějící podpora trojrozměrného prostoru, proto knihovna není použita v aplikaci.

4.1.2 Stochastic Simulation in Java (SSJ)

Knihovna pro náhodné simulace, vyvinuta v Université de Montréal. Poskytuje prostředky pro generování uniformních a neuniformních náhodných variací, výpočet rozdílných mír týkajících se rozdělení pravděpodobnosti, provádění testů dobré shody, aplikaci Kvazi-Monte Carlo metody, sbírání základních statistik a programování diskrétních simulací událostí s eventy a procesy. [13]

Knihovna je v aplikaci využita pro výpočet průběhů polynomů a aproximaci metodou nejmenších čtverců. Většina knihovny je tedy nevyužita. Knihovna je poskytovaná pod licencí GPL, proto i celá aplikace musí být poskytována pod licencí GPL. Protože se jedná o bakalářskou práci, není tato licence problémem.

4.1.3 Colt

Colt poskytuje sadu knihoven pro vědecké a technické výpočty v Javě. [14]

Aplikace tuto knihovnu využívá pro maticové operace, použité pro výpočet vykreslovaných křivek a ploch. Dle licence knihovny musí být v technické dokumentaci aplikací využívající knihovnu uveden text:

Copyright (c) 1999 CERN – European Organization for Nuclear Research.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. CERN makes no representations about the suitability of this software for any purpose. It is provided "as is" without expressed or implied warranty.

4.1.4 Geometry library (JGEOM)

Knihovna pro výpočet NURBS křivek a ploch, umožňující provádět s těmito objekty logické operace (například průnik). [15]

Knihovna není v aplikaci použita, protože jsem se rozhodl pro vlastní implementaci NURBS, dalším důvodem je, že v některé z budoucích verzí knihovny JOGL bude zahrnuta podpora NURBS objektu přímo s hardwarovou akcelerací pomocí grafické karty.

4.1.5 JOGL

Knihovna umožňující přístup k API OpenGL knihovny. Poskytuje přístup k hardwarově akcelerovaným 3D aplikacím v Javě. [16]

Tato knihovna je stěžejní částí aplikace, stará se o grafický výstup a některé křivky a plochy v aplikaci umožňují zapnout akceleraci výpočtu pomocí grafické karty. V aktuální verzi knihovna nepodporuje objekt NURBS.

4.1.6 Java 3D

Java 3D slouží k zobrazování trojrozměrné grafiky. Programy napsané v Java 3D mohou běžet na mnoha odlišných typech počítačů a přes internet. Java 3D staví na existujících technologiích jako je DirectX a OpenGL. [17]

Tato knihovna je konkurencí knihovny JOGL, rozhodl jsem se použít knihovnu JOGL, protože její API je velice podobné API knihovny OpenGL v jazyce C.

4.1.7 HEJPBézier

Další knihovnou je jednoúčelová knihovna HEJPBézier pro výpočet Bézierovy křivky. Knihovna podporuje výpočet Bézierových křivek různých stupňů [18]. Knihovna obsahuje pouze jedinou třídu.

4.1.8 JAMA (A Java Matrix Package)

Knihovna implementující operace lineární algebry. Poskytuje funkce pro konstruování a manipulování s reálnými maticemi. Implementuje elementární operace s maticemi a algoritmy pro LU rozklad, QR rozklad, Choleského rozklad, singulární rozklad a vlastní rozklad matic [19].

4.1.9 Efficient Java Matrix Library (EJML)

Knihovna pro manipulaci a výpočty s hustými maticemi. Knihovna má být výpočetně a paměťově úsporná pro malé i velké matice, a má být přístupná, jak pro začátečníky, tak i pro experty. Tyto cíle jsou splněny tím, že dynamicky vybírá nejlepší algoritmy za běhu, a návrhem čistého API. EJML je zdarma a je uvolněna pod licencí LGPL [20].

4.1.10 Batik

Batika je knihovna pro práci se škálovatelnou vektorovou grafikou (SVG). Umožňuje manipulovat, zobrazovat a vytvářet SVG dokumenty a exportovat je do jiných formátů [21]. Knihovna je uvolněna pod Apache licencí. Tato knihovna by mohla být použita v některé z budoucích verzí aplikace pro exportování vytvořených křivek a ploch do formátu SVG.

4.2 Použité knihovny

Aplikace využívá knihovnu JOGL pro vykreslování křivek a ploch, knihovnu SSJ pro jednodušší práci s polynomy a knihovnu COLT pro operace s maticemi. Díky multiplatformnosti jazyku Java, je výsledná aplikace spustitelná na různých operačních systémech, byla otestována na operačním systému Linux (Ubuntu 10.10) a Microsoft Windows 7.

5 Analýza a návrh aplikace

Cílem návrhu aplikace je navrhnout takovou aplikaci, která bude splňovat všechny kladené požadavky. Návrh aplikace slouží jako základ pro její implementaci. Protože aplikace se bude skládat z většího množství tříd, budou rozděleny do balíčků. K modelování byla použita volně dostupná aplikace ArgoUML, která je dostupná na adrese [22].

Aplikace je navržena podle návrhového vzoru MVC tak, aby byla snadno rozšiřitelná o nové křivky a plochy. Model se stará o operace s daty a jejich ukládání. Pohled se stará o zobrazování informací a o uživatelský vstup, který prostřednictvím událostí oznamuje posluchačům, posluchači patří do části kontroléru, kteří události zpracovávají a řídí tak běh programu. Důležitou vlastností vzoru je, že model neví nic o pohledu a kontroléru, ale pohled musí vědět o modelu, ze kterého získává data. Kontrolér musí vědět o modelu i pohledu, zachytává události uživatelského vstupu pohledu a nastavuje pohled i model. Více o návrhovém vzoru MVC můžete nalézt například v [23].

5.1 Rozdělení na balíčky

Celá aplikace je umístěna v balíčku **cscg**, tento balíček obsahuje hlavní spustitelnou třídu a tři balíčky: **model**, **controller** a **ui**, tyto balíčky přímo představují rozdělení aplikace podle vzoru MVC. Na obrázku 5.1 je zobrazen diagram balíčků.

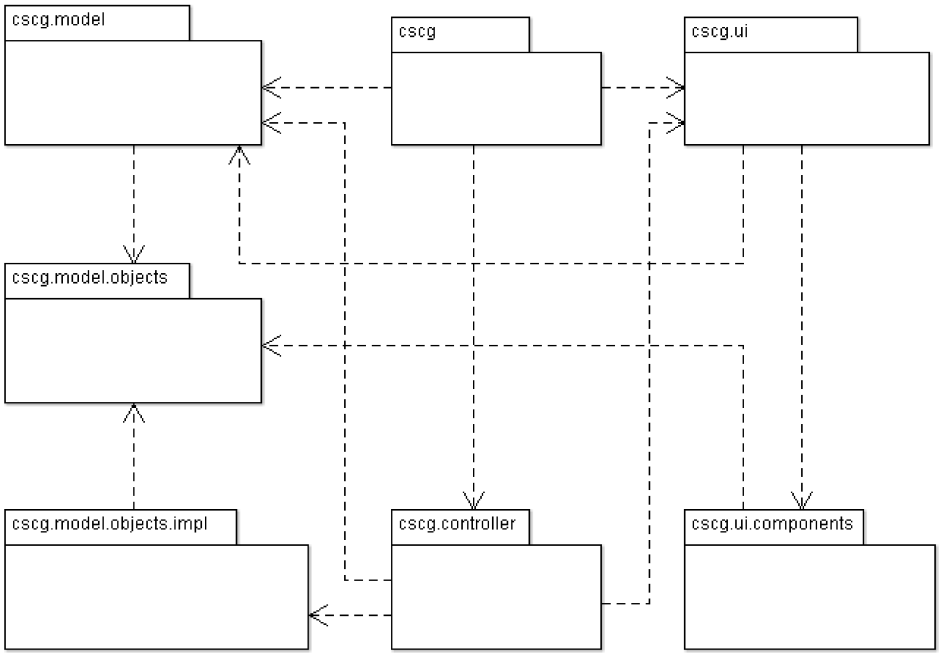
V kapitole je uveden popis jednotlivých balíčků balíčků aplikace. Informace o jednotlivých třídách aplikace naleznete v API dokumentaci v příloze [F].

5.1.1 Balíček cscg

Hlavní balíček. Obsahuje všechny balíčky aplikace a spustitelnou třídu **Cscg**, která obsahuje statickou metodu **main**, ve které se vytvoří instance kontroléru **controller.Controller**, pohledu **ui.View** a modelu **model.Model** a následně se předá řízení kontroléru.

5.1.2 Balíček cscg.model

Balíček obsahující třídy zařaditelné podle návrhového vzoru MVC do části modelu. Aby se aplikace spustila vždy ve stavu v jakém byla ukončena, je hlavní třída modelu **Main** podle návrhového vzoru singleton a vždy po spuštění aplikace je načtena serializovaná podoba instance třídy z pracovního adresáře aplikace a při ukončení aplikace je instance opět serializována. Na obrázku v příloze [C] je zobrazen diagram tříd balíčku.



Obrázek 5.1: Diagram balíčků

5.1.3 Balíček cscg.model.objects

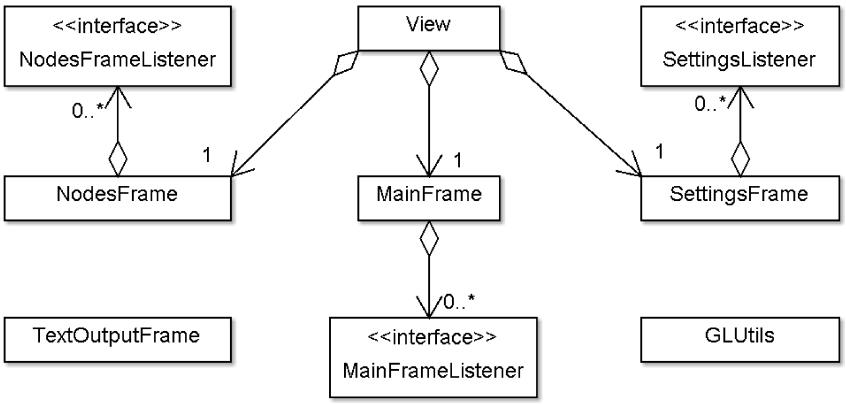
Balíček obsahující třídy a rozhraní pro křivky v projektu. Objektem se rozumí zobrazitelné prvky projektu, které mohou být například křivka nebo plocha. Návrh balíčku je takový, že všechny objekty musí implementovat rozhraní **IObject**, který obsahuje předpisy základních metod pro práci s objekty – nastavení řídících bodů, rotaci, posun, viditelnost objektu a další. Na obrázku v příloze [D] je zobrazen diagram tříd balíčku.

5.1.4 Balíček cscg.model.objects.impl

Balíček rozšiřující aplikaci o konkrétní implementace křivek a ploch. Třídy tohoto balíčku můžou sloužit jako vzor při rozšiřování aplikace o další typy křivek a ploch v externí knihovně. Kromě tříd implementovaných křivek a ploch obsahuje balíček třídy **Point3fChangeable** a **Point4fChangeable**, které jsou rozšířením tříd **cscg.model.objects.Point3f** a **cscg.model.objects.Point4f** o chráněné metody pro změnu souřadnic. Využívají je implementované křivky a plochy, které umožňují nastavit spojitost napojování křivek, kdy je potřeba změnit polohu bodu podle zvoleného napojování (například Bézierova křivka). Na obrázku v příloze [E] je zobrazen diagram tříd balíčku **cscg.model.objects.impl**, diagram zobrazuje i vybrané třídy z balíčku **cscg.model.objects**, aby byly zdůrazněny návaznosti. Zeleně jsou vyznačeny třídy, které implementují křivky a plochy.

5.1.5 Balíček cscg.ui

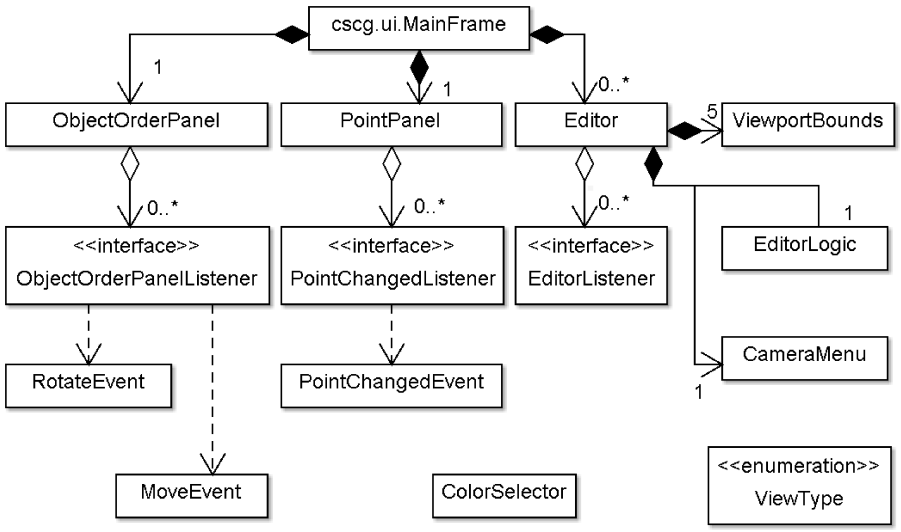
Balíček tříd pro vytvoření uživatelského rozhraní. Balíček obsahuje třídy zařaditelné podle návrhového vzoru MVC do části pohledu. Celý pohled se skládá ze tří oken – hlavního okna programu MainFrame, okna pro úpravu řídících bodů a uzlových vektorů objektů a okna pro nastavení aplikace. Všechny události oken vzniklé na základě uživatelského vstupu jsou prostřednictvím událostí předány kontroléru, který se stará o jejich obsluhu. Na 5.2 obrázku je zobrazen diagram tříd balíčku.



Obrázek 5.2: Diagram tříd balíčku cscg.ui

5.1.6 Balíček cscg.ui.components

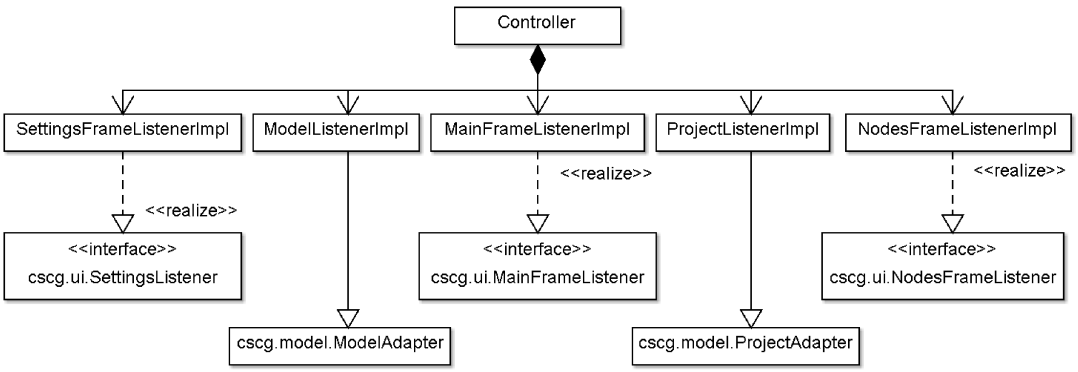
Balíček obsahující třídy specializovaných GUI komponent využitých především v okně **MainFrame**. V následujícím diagramu 5.3 je pro přehlednost zobrazena i třída **cscg.ui.MainFrame**. Balíček obsahuje GUI komponenty pro výběr barvy, nastavení vlastností řídícího bodu, seznam objektů v projektu a panel vykreslující samotné křivky a plochy.



Obrázek 5.3: Diagram tříd balíčku cscg.ui.components

5.1.7 Balíček cscg.controller

Balíček tříd starající se o běh programu. Balíček obsahující třídy zařaditelné podle návrhového vzoru MVC do části kontroléru. V diagramu tříd 5.4 jsou zobrazeny i některé třídy z balíčků **cscg.ui** a **cscg.model** pro zachycení souvislostí. Hlavní třída kontroléru **Controller** se stará o vytvoření posluchačů jednotlivých oken pohledu.



Obrázek 5.4: Diagram tříd balíčku cscg.controller

6 Implementace

Výsledkem fáze implementace je funkční aplikace implementovaná podle jejího návrhu. V rámci implementace bylo potřeba vyřešit různé matematické algoritmy, problémy se spouštěním aplikace na různých operačních systémech a optimalizovat rozvržení GUI. K aplikaci byla vytvořena API dokumentace, která je přílohou bakalářské práce [F].

6.1 Spouštění na různých operačních systémech

Knihovna JOGL ke své funkci potřebuje binární knihovny, které jsou rozdílné pro každý operační systém. Aby bylo možné aplikaci spouštět z jedné složky, byly umístěny všechny binární knihovny do podsložek v příloze [H]. Při spuštění hlavní aplikace (cscg.jar) se musí nastavit cesta k binárním knihovnám, aby se to nemuselo dělat ručně při každém spuštění, byla vytvořena další spouštěcí aplikace, která spustí hlavní aplikaci s příslušnými parametry, projekt pomocné aplikace je v příloze [I]. Návod ke spuštění aplikace je v příloze [A].

6.2 GUI

Celá aplikace je rozdělena do tři oken. O tyto okna se stará třída **cscg.ui.View**. Obrázky všech oken jsou v příloze [A].

Hlavní okno třídy **cscg.ui.MainFrame** je rozděleno na několik částí:

- Menu: umožňuje práci s projekty (vytvářet, otevírat, zavírat, exportovat), nastavovat způsob zobrazení projektu a přidávat objekty (křivky a plochy) do projektu.
- Panel nastavení objektu: umožňuje nastavovat vlastnosti vybraného objektu projektu. Tento panel si každý objekt vytváří sám, jeho vzhled záleží na vybraném objektu, umožňuje například nastavit stupeň křivky nebo plochy, barvu, jméno, počty bodů řídící sítě ploch, přesnost vykreslení atd.
- Panel seznamu vytvořených objektů v projektu: umožňuje měnit pořadí objektů, nastavovat viditelnost objektů, mazat objekty, posunovat a rotovat objekty. Je instance třídy cscg.ui.components.ObjectOrderPanel.
- Panel nastavení bodu: instance třídy cscg.ui.components.PointPanel, zobrazuje a umožňuje měnit souřadnice vybraného řídicího bodu, popřípadě váhu bodu. V panelu se zobrazuje i GUI vytvořené vybraným objektem, například u Hermitovy křivky se zobrazuje směrnice tečny řídicích bodů.
- Editor projektu: vlastní OpenGL okno umožňující editovat a vykreslovat vybraný projekt. Editor je instancí stejnojmenné třídy cscg.ui.components.Editor. Každý projekt

má vytvořenu vlastní instanci třídy Editor, jednotlivé editory jsou umístěny a zobrazeny v záložkách.

Další okno je okno s nastavením programu, je instancí třídy **cscg.ui.SettingsFrame**. Umožňuje nastavit kvalitu vykreslování, a vlastnosti osvětlování objektů.

Posledním oknem je okno třídy **cscg.ui.NodesFrame** pro zobrazení a editaci řídících bodů vybrané křivky nebo plochy. Okno je rozděleno na dvě části: vrchní a spodní. Vrchní část je zobrazena pouze pokud objekt implementuje rozhraní cscg.model.objects.INonUniformCurve nebo cscg.model.objects.INonUniformSurface. Umožňuje editovat uzlové vektory. Uzlový vektor je zobrazen v tabulce v řádcích po patnácti sloupcích. Řádky tabulky se přidávají postupně s rostoucí velikostí uzlového vektoru. Spodní část okna zobrazuje tabulku řídicích bodů objektu. V prvním sloupci je zobrazeno pořadí bodu, pokud objekt implementuje rozhraní cscg.model.objects.ISurface, pak je zobrazena souřadnice v mřížce. Další sloupce zobrazují *x, y, z* souřadnice. Pokud jsou body instance cscg.model.objects.IPoint4f, pak je zobrazen sloupec s vahou bodu. Tabulka řídících bodů i tabulky uzlových vektorů umožňují editovat zobrazené hodnoty.

6.3 Editor (OpenGL panel)

Editor je třída, vytvářející grafické rozhraní, které umožňuje vykreslovat a upravovat projekty. Třída je potomkem třídy **GLJPanel** z knihovny JOGL. K vykreslení grafického výstupu editoru je nutné vytvořit instanci třídy **FPSAnimator**, která se stará o periodické vykreslování obsahu.

Pro každý projekt je vytvořena jedna instance třídy **Editor**. O vytváření instancí tříd Editor a FPSAnimator se stará hlavní okno aplikace, respektive třída **MainFrame**. Protože v aplikaci můžeme v jeden okamžik mít otevřen větší počet projektů, je vytvořena instance třídy **FPSAnimator** pro každý projekt.

Pravidelné vykreslování editoru je časově náročná operace, především u projektů, které obsahují složité objekty (plochy s vysokým stupněm přesnosti vykreslení), proto je v jeden okamžik aktivní pouze **FPSAnimator** aktuálně zobrazeného projektu a ostatní animátory jsou pozastaveny.

Třída **Editor** je úzce spojena s třídou **EditorLogic**, která zajišťuje vlastní funkčnost editoru. Třída implementuje rozhraní **GLEventListener** (z knihovny JOGL). Z tohoto rozhraní musí implementovat metody init, reshape, display.

Metoda init je volána pouze před prvním vykreslením (na systémech Linux je volána i po každé změně rozměrů okna). V této metodě jsou inicializovány některé proměnné závislé

na aktuálním OpenGL kontextu (OpenGL kontext je aktuální stav OpenGL stavového automatu, každé vlákno může mít jiný kontext). Jsou nahrány obrázky z pevného disku do paměti grafické karty, jsou vytvořeny instance tříd **GLU** (OpenGL Utility Library) a **GLUT** (The OpenGL Utility Toolkit).

Metoda reshape je volána pouze v případě, když dojde ke změně rozměrů okna. Jediná akce, která je v metodě provedena, je uložení nových rozměrů okna a výpočet rozvržení plochy editoru na čtyři stejně velké obdélníky pohledů (každý pohled může mít nastavenu vlastní projekci a zobrazen jiný grafický obsah).

Metoda display se stará o samotné vykreslení, tuto metodu pravidelně volá **FPSAnimator**. Metoda první vytvoří pro každý objekt projektu display list, pokud je nastaven příznak, informující o zneplatnění nebo pokud display list neexistuje. Poté vykreslí čtyři nebo jeden (podle nastavení projektu) pohled, pro každý pohled nastaví projekci, rámeček, informační popisky a nakonec vykreslí v pořadí všechny objekty projektu, které jsou v paměti přichystané v podobě display listů. Nakonec projde frontu událostí myši, a provede akce přiřazené k událostem.

Události myši se musí obsluhovat v těle metody display, protože některé akce myši, například posun objektu, potřebují ke svému výpočtu číst z hloubkového bufferu grafické karty. Pro frontu událostí je použit seznam třídy **LinkedBlockingQueue** z balíčku java.util.concurrent, který je optimalizovaný pro vícevláknový přístup (první vlákno je vlákno animátoru, které odebírá události, druhé vlákno je vlákno vytvořené knihovnou SWING, starající se o události v GUI objektech).

6.4 Implementace vybraných matematických úloh

Protože některé operace s body a vektory jsou využívány ve více třídách a jsou obecné, byla vytvořena třída **cscg.model.object.PointOperations**, která operace implementuje.

6.4.1 Implementované operace ve třídě PointOperations

Rotace vektoru kolem osy - metoda axisRotationVector3

Metoda využívá Rodriguesovi věty o rotaci [24]:

$$\vec{v}_{rot} = \vec{v}\cos\theta + (\vec{k}\times\vec{v})\sin\theta + \vec{k}(\vec{k}\cdot\vec{v})(1-\cos\theta) \tag{6.1}$$

Vstup: původní vektor $\vec{v}$, úhel rotace θ, osa rotace $\vec{k}$.
Výstup: rotovaný vektor $\vec{v}_{rot}$.

Porovnání dvou bodů - metoda compareCoords

$$I_x = J_x \wedge I_y = J_y \wedge I_z = J_z \tag{6.2}$$

Vstup: bod *I* a bod *J*.
Výstup: body si (ne)jsou rovny.

Výpočet směrového vektoru z jednoho bodu do druhého – metoda directionVector

$$\vec{v} = J - I \tag{6.3}$$

Vstup: bod *I* a bod *J*.
Výstup: vektor $\vec{v}$ z bodu *I* do bodu *J*.

Výpočet vzdálenosti jednoho bodu k druhému – metoda distance

Metoda využívá dvou dalších metod – directionVector a sizeVector3. Vzdálenost vypočte jako velikost směrového vektoru.

$$d = sizeVector3(directionVector(I, J)) \tag{6.4}$$

Vstup: bod *I* a bod *J*.
Výstup: vzdálenost zadaných bodů *d*.

Výpočet nejbližšího bodu na přímce k zadanému bodu – metoda intersection

Vstup: přímka *p* zadaná body *P1* a *P2*, a bod *J*.
Výstup: nejbližší bod *X* přímky *p* k bodu *J*.
Algoritmus se skládá z více kroků:

1. Pokud $P_1 = P_2$ pak se nejedná o přímku, ale o bod a výsledkem je bod P_1.
2. Pokud bod *J* leží přímo na přímce (zjištění pomocí metody isLine), pak je výsledkem bod *J*.
3. Nejbližší bod je vypočítán jako průsečík přímky *p* a roviny α kolmé k přímce, ve které leží bod *J*.
4. Vyjádření parametrické rovnice přímky *p*:

$$\begin{aligned} \vec{d} &= P_1 - P_2 \\ x &= P_{1x} + t \cdot \vec{d}_x \\ y &= P_{1y} + t \cdot \vec{d}_y \\ z &= P_{1z} + t \cdot \vec{d}_z \end{aligned} \tag{6.5}$$

5. Vytvoření obecné rovnice roviny α. Normálový vektor roviny je roven směrovému vektoru přímky $\vec{d}$, bod *J* leží v rovině.

$$\vec{d}_x(x - J_x) + \vec{d}_y(y - J_y) + \vec{d}_z(z - J_z) = 0 \tag{6.6}$$

6. Dosazení parametrické rovnice přímky 6.5 do rovnice roviny 6.6 a vyjádření parametru *t*.

$$t = \frac{[\vec{d}_x \cdot (-P_{1x} + J_x)] + [\vec{d}_y \cdot (-P_{1y} + J_y)] + [\vec{d}_z \cdot (-P_{1z} + J_z)]}{\vec{d}_x^2 + \vec{d}_y^2 + \vec{d}_z^2} \tag{6.7}$$

7. Dosazením vypočítaného parametru *t* do rovnice přímky 6.5 vypočítáme průsečík roviny s přímkou, který je zároveň i hledaný bod.

Zjištění jestli tři body leží na společné přímce – metoda isLine

Vstup: body *I, J, K*.
Výstup: body (ne)leží na přímce.

1. Pokud jsou si alespoň dva body rovny, pak všechny body leží na přímce.
2. Určení parametrické rovnice přímky dané body *I* a *J*.

$$\begin{aligned} \vec{d} &= I - J \\ x &= I_x + t \cdot \vec{d}_x \\ y &= I_y + t \cdot \vec{d}_y \\ z &= I_z + t \cdot \vec{d}_z \end{aligned} \tag{6.8}$$

3. Do rovnice přímky je dosazen bod *K*, pokud má soustava řešení, pak body leží na přímce.

Výpočet obrazu bodu podle středové souměrnosti – metoda mirror

Vstup: bod *P* a střed souměrnosti *S*.
Výstup: obraz bodu P_S.

$$P_S = \left[2 \cdot S_x - P_x \; ; \; 2 \cdot S_y - P_y \; ; \; 2 \cdot S_z - P_z \right] \tag{6.9}$$

Normalizace vektoru – metoda normalizeVector3

Vstup: vektor $\vec{v}$.
Výstup: normalizovaný vektor $\vec{v}_n$.

$$\begin{aligned} s &= \|\vec{v}\| \\ \vec{v}_n &= \left[\frac{\vec{v}_x}{s} ; \frac{\vec{v}_y}{s} ; \frac{\vec{v}_z}{s} \right] \end{aligned} \tag{6.10}$$

Výpočet středu úsečky zadané dvěma body – metoda pivot

Vstup: body *I, J*.
Výstup: střed úsečky *S*.

$$S=\left[\frac{I_x+J_x}{2};\frac{I_y+J_y}{2};\frac{I_z+J_z}{2}\right] \tag{6.11}$$

Výpočet velikosti vektoru – metoda size

Vstup: vektor $\vec{v}$.
Výstup: velikost vektoru $\|\vec{v}\|$.

$$\|\vec{v}\|=\sqrt{\vec{v}_x^2+\vec{v}_y^2+\vec{v}_z^2} \tag{6.12}$$

6.5 Implementace metod nad OpenGL

Při vykreslování křivek, ploch a uživatelského rozhraní pomocí OpenGL se některé operace často opakují a jsou znovupoužitelné v různých třídách. Proto byla implementována třída cscg.ui.GLUtils, která tyto operace obsahuje.

6.5.1 Třída GLUtils

Třída **GLUtils** obsahuje statické metody často prováděných operacích nad OpenGL nebo komplexních operací pro vykreslení různých objektů. V této kapitole budou popsány pouze vybrané funkce, informace o ostatních funkcích můžete najít v API dokumentaci v příloze [F].

Vykreslení šipky – metoda drawArrow

Funkce vykreslí 2D šipku, která se správně zobrazí pro libovolnou projekci v 3D prostoru. Šipka je zadaná počátečním a koncovým bodem, poloměrem *r* podstavy trojúhelníku a výškou *v* trojúhelníku tvořícího hrot šipky. Šipka je vykreslena pomocí přímky spojující počáteční a koncový bod a pomocí rovnoramenného trojúhelníku. Vrchol protilehlý přeponě trojúhelníku je roven koncovému bodu šipky. Algoritmus pro vykreslení šipky funguje na principu výpočtu šipky v 2D prostoru v ploše zobrazené na obrazovce a převodu vypočítané šipky do 3D souřadnic OpenGL:

1. Počáteční a koncový bod šipky v 3D souřadnicích OpenGL je převeden na 2D souřadnice na obrazovce.
2. V 2D souřadnicích je vypočítán úhel natočení šipky.
3. V 2D souřadnicích jsou vypočítány souřadnice rovnoramenného trojúhelníku (hrotu šipky) který má přeponu rovnoběžnou s osou *x*, délka přepony je rovna 2 x *r*, délka výšky na přeponu je rovna *v*. Vrchol protilehlý přeponě je roven bodu konce šipky.
4. Vypočítané souřadnice trojúhelníku jsou převedeny do 3D souřadnic OpenGL.
5. Je vykreslena čára mezi koncovým a počátečním bodem šipky.

6. Souřadnice trojúhelníku šipky v 3D prostoru jsou rotovány o úhel vypočtený v bodě 2). Tento krok je nutný protože trojúhelník šipky byl v bodě 3) vypočítán s přeponou rovnoběžnou s osou *x*.
7. Vykreslení vypočítaného trojúhelníku.

Vykreslení ikony orientace kamery v prostoru – metoda drawOrientationCameraIcon

Vykreslení ikony šipky, která mění svoji orientaci v prostoru podle nastavení projekce zadané v parametru funkce. Funkce nejdříve uloží aktuální matici projekce, poté nastaví kameru na směr stejný podle zadané projekce, vykreslí 3D šipku skládající se z kvádru a kužele a nakonec obnoví původní nastavení projekce.

Vykreslení textu – metoda drawTextBox

Vykreslení textu, zadaného frontou řádků, do obdélníkové oblasti, s volbou horizontálního zarovnání (vpravo/vlevo) a vertikálního zarovnání (umístění boxu směrem nahoru/dolů od aktuální pozice). Umožňuje nastavit velikost řádku a mezeru mezi řádky.

Vykreslení plochy zadané mřížkou bodů – metoda drawSurfaceViaTriangles

Vykreslení plochy zadané mřížkou bodů. Plocha je vykreslena jako množství spojených trojúhelníků. Pro každý vrchol plochy je vypočítána normála (v OpenGL mají normály i body!). Normála je vypočítána pomocí více vláken, počet vláken je určen počtem dostupných procesorů. Normála je vypočítána následujícím postupem:

1. Každý bod má šest sousedních bodů (body u okrajů plochy méně), sousední body si označíme P_0 až P_5, bod uprostřed si označíme P. Do každého z těchto sousedních bodů je veden směrový vektor, ty si označíme $\vec{v}_0$ až $\vec{v}_5$.

$$\begin{aligned}\vec{v}_0 &= P_0 - P\\ \vec{v}_1 &= P_1 - P\\ \vec{v}_2 &= P_2 - P\\ \vec{v}_3 &= P_3 - P\\ \vec{v}_4 &= P_4 - P\\ \vec{v}_5 &= P_5 - P\end{aligned} \tag{6.13}$$

2. Pokud některý ze sousedních bodů chybí, je směrový vektor nahrazen nulovým vektorem.
3. Postupně proti směru hodinových ručiček je proveden vždy vektorový součin aktuálního vektoru s následujícím. Tímto získáme normály všech trojúhelníků plochy, které mají jeden vrchol v bodě *P*.

$$\begin{aligned}\vec{n}_5 &= \vec{v}_5 \times \vec{v}_4 \\ \vec{n}_4 &= \vec{v}_4 \times \vec{v}_3 \\ \vec{n}_3 &= \vec{v}_3 \times \vec{v}_2 \\ \vec{n}_2 &= \vec{v}_2 \times \vec{v}_1 \\ \vec{n}_1 &= \vec{v}_1 \times \vec{v}_0 \\ \vec{n}_0 &= \vec{v}_0 \times \vec{v}_5\end{aligned} \tag{6.14}$$

4. Všechny výsledky vektorových součinů sečteme, tím získáme normálu v daném bodě *P*, označíme si ji $\vec{n}$.

$$\vec{n} = \sum_{i=0}^{5} \vec{n}_i \tag{6.15}$$

6.6 Nastavení projekce

Každé OpenGL okno může zobrazit libovolný počet pohledů (viewportů), každý pohled má svou vlastní projekční a modelovou matici a může vykreslovat vlastní scénu. Nastavení těchto dvou matic ovlivňuje jaká část scény se vykreslí. Třída **cscg.ui.Editor** využívá pro nastavení projekční a modelové matice třídu **cscg.model.Projection**, která přestavuje nastavení projekce a implementuje operace potřebné pro její aplikování a změnu.

Projekční matice slouží k nastavení vlastností záběru scény. Matice umožňuje nastavit tvar výřezu prostoru, který je kamerou snímaný. Tvar může být kvádr pro ortogonální projekci nebo komolý jehlan pro projekci perspektivní. Modelová matice se používá ke změně směru polohy a natočení kamery. Pomocí této matice lze také definovat měřítko zobrazení [25].

Třída **Projection** slouží k uložení všech vlastností projekce: polohy kamery, natočení kamery, měřítko a volbu mezi perspektivní a ortogonální projekcí.

6.6.1 Metody pro nastavení vlastností projekce

V této kapitole jsou popsány vybrané operace pro nastavení projekce scény.

Nastavení pohledu

Nastavení pohledu zepředu, zprava a shora pomocí metod **lookFront**, **lookProfile** a **lookTop**. Metody přesunou pozici kamery do počátku a změní směr natočení kamery.

Změna polohy kamery – metoda move

Změna polohy kamery bez změny jejího natočení. Existuje několik variant této metody:

1. Zadáním zdrojového bodu v prostoru, který chceme přesunout na pozici cílového bodu v prostoru. Metodu využívá například obsluha událostí myši, která pomocí metody **GLUtils.gluUnProject** zjistí body v prostoru odkud a kam bylo taženo myší. Metoda zjistí směrový vektor $\vec{v}$ zadaných bodů P_0 a P_1 daný vektor přičte k aktuální pozici kamery C_0. Nová pozice kamery je označena C_1.

$$\begin{aligned} \vec{v} &= P_1 - P_0 \\ C_1 &= C_0 + \vec{v} \end{aligned} \tag{6.16}$$

2. Zadáním vzdálenosti o kterou se má posunout pozice kamery ve směru vpřed označená *f*, vpravo označená *r* a nahoru označená *u*. Třída Projection si udržuje v každém okamžiku tři vektory: $\vec{d}$ vektor, který udává směr kam je kamera natočena, $\vec{u}$ vektor, který udává směr, kde se nachází vrch projekce (respektive stranu, která se v okně vykreslí nahoře) a $\vec{r}$ vektor, který udává, která strana je vpravo. Všechny tři vektory jsou navzájem na sebe kolmé a jsou jednotkové. Tato metoda se využívá pro posun kamery pomocí tlačítek klávesnice. Aktuální pozice kamery je označena C_0, nová pozice kamery je označena C_1.

$$C_1 = C_0 + f \cdot \vec{d} + r \cdot \vec{r} + u \cdot \vec{u} \tag{6.17}$$

3. Zadáním vektoru posunu $\vec{v}$. Aktuální pozice kamery je označena C_0, nová pozice kamery je označena C_1.

$$C_1 = C_0 + \vec{v} \tag{6.18}$$

Rotace kamery – metoda rotate

Změna směru natočení kamery. Metoda má několik variant:

1. Rotace zadáním úhlů rotace kolem vertikální, horizontální osy a kolem osy ve směru pohledu kamery.
2. Rotace zadáním úhlu a osy rotace pomocí bodu a vektoru.
3. Předchozí dvě varianty navíc se zadáním fixního bodu, který určí bod na obrazovce, který má zůstat na svém místě. Například pokud zadáme jako fixní bod [10;20;30], který se před rotací nachází v pravém horním rohu obrazovky, bude se po provedení rotace nacházet co nejblíže pravému hornímu rohu obrazovky. Tato metoda má využití při rotaci kamery pomocí myši, kdy se fixuje bod nacházející se uprostřed vybraného objektu projektu a rotace se uživateli jeví tak, jakoby rotoval s vybraným objektem.
   Výpočet rotace kamery:
   1. Vektory $\vec{d}$, $\vec{u}$, $\vec{r}$ udávají aktuální směr natočení kamery směrem vpřed, nahoru a vpravo. Tyto vektory jsou jednotkové a jsou navzájem kolmé. Výsledkem rotace bude jejich změna.
   2. Vektory jsou rotovány pomocí metody **PointOperations.axisRotationVector3**.

3. Vektory jsou normalizovány na jednotkovou délku pomocí funkce **PointOperations.normalizeVector3**.

Pokud rotujeme včetně zadání fixního bodu, pak se před předchozí postup přidají následující kroky:

1. Je vypočítán vektor $\vec{t}$ z fixního bodu do pozice kamery.
2. Vektor $\vec{t}$ je rotován podle zadané rotace pomocí funkce **PointOperations.axisRotationVector3**.
3. Pozice kamery je posunuta o vektor $\vec{t}$.
4. Dále je kamera rotována běžným způsobem.

6.7 Rozšíření aplikace o další typy křivek a ploch

Rozšířit aplikaci o další typy křivek a ploch můžeme dvěma způsoby: buď upravíme aplikaci samotnou nebo vytvoříme novou knihovnu, kterou k aplikaci připojíme. Pro usnadnění rozšíření aplikace byla vytvořena API dokumentace v příloze [F] a návod, v příloze [B], s uvedeným postupem, včetně uvedených příkladů.

Pro usnadnění vývoje aplikace, byl vytvořen veřejný GIT repositář na adrese https://github.com/freezzone/cscg. Git je systém správy verzí a web http://github.com poskytuje zdarma prostor pro umístění repositářů aplikací vyvíjených s otevřeným zdrojovým kódem.

7 Testování aplikace

Důležitou součástí vývoje aplikace je její průběžné testování. Cílem každého průběžného testování je otestovat správnou funkčnost již implementovaných komponent aplikace a odhalit jejich chyby. Aplikace byla testovaná na následujících počítačových sestavách:

1. CPU: AMD Phenom II X4 955 3,2GHz
   RAM: 4GB
   MB: Gigabyte GA-MA770T-UD3P
   GPU: ATI Radeon HD 5700
   OS: Windows 7 Professional SP1 64-bit, Ubuntu 10.10 64-bit

2. CPU: Intel Mobile Core 2 Duo P9300 2,66GHz
   RAM: 2GB
   MB: Hawlett-Packard
   GPU: Mobile Intel 4 Series Express
   OS: Windows 7 Professional SP1 32-bit

Při testování se projevily problémy s vykreslováním na některých grafických kartách, kdy některé grafické karty vyhlazovaly okraje polygonů, přestože v aplikaci bylo vyhlazování zakázané. To způsobovalo nepěkné vykreslení ploch, které se vykreslily nejednolitě a docházelo k velkému poklesu rychlosti vykreslení, aplikace se stávala nepoužitelná. Byl hledán důvod této chyby, nakonec bylo zjištěno, že se jednalo o chybu starší verze knihovny JOGL. Nejnovější verze knihovny touto chybou netrpí, ale nepodporuje oproti starší verzi hardwarovou akceleraci vykreslení NURBS objektů, proto nakonec aplikace podporuje výpočet NURBS objektů pouze pomocí procesoru.

Dalším problémem bylo zobrazení menu přes GLCanvas komponentu (komponenta vykreslující obsah pomocí OpenGL). GLCanvas se vykresloval téměř vždy nahoře a položky menu nebyly vidět. Problém se podařilo odstranit nahrazením komponenty GLCanvas komponentou GLJpanel, která vykresluje obsah pomocí OpenGL první do bufferu a teprve poté výsledný obraz zobrazí.

Na operačním systému Ubuntu byl problém s přilinkováním binárních knihoven JOGL. Ubuntu při spouštění aplikace nenastaví, na rozdíl od OS Windows, automaticky cestu pro hledání binárních knihoven na adresář, ve kterém se nachází aplikace (nejedná se o chybu systému Ubuntu, ale o vlastnost). Problém byl vyřešený pomocí vytvoření druhé aplikace , která spouští aplikaci hlavní, více v kapitole 6.1.

Bylo provedeno otestování rychlosti aplikace, cílem testu bylo zjistit jak složitou plochu lze v aplikaci vykreslit, tak aby výpočet a vykreslení plochy netrvalo déle než jedna vteřina. V aplikaci byly postupně vytvořeny Bézierova, Coonsova a NURBS plocha a byly zvětšovány řídící sítě bodů tak dlouho, dokud byl naměřený čas kratší než jedna vteřina. Čas byl měřen v aplikaci, která byla rozšířena, tak aby vypisovala informace o době výpočtu a vykreslení objektu do konzoly. Tabulka změřených časů:

Plocha/Testovací sestava	**Sestava 1**	**Sestava 2**
Bézierova	Síť 81 x 81	Síť 58 x 58
Coonsova	Síť 33 x 33	Síť 27 x 27
NURBS	Síť 38 x 38	Síť 28 x 28

Všechny plochy byly vykresleny s volbou automatické přesnosti. Bézierovy a NURBS plochy měli nastavený třetí stupeň plochy, u Bézierovy plochy byla povolena akcelerace pomocí grafické karty.

Rychlost vykreslení ploch hodnotím jako dostačující pro účely výuky. Pokud by uživatel potřeboval vykreslit plochy zadané více řídícími body, může si snížit přesnost vykreslení.

Byla otestována rychlost vykreslení křivek, implementované křivky mají mnohonásobně vyšší rychlost výpočtu a vykreslení než implementované plochy, proto byly otestovány jiným způsobem. Každá křivka byla tvořena 40 řídícími body a byl měřen čas jejich výpočtu a vykreslení.

Křivka/Testovací sestava	**Sestava 1 [ms]**	**Sestava 2 [ms]**
Interpolace polynomem	3	1
Hermitova interpolace	2	1
Fergusonova křivka	2	1
Aproximace metodou nejmenších čtverců (stupeň křivky = 39)	2	1
Bézierova křivka (stupeň křivky = 39)	14	16
Coonsův B-spline	2	1
NURBS křivka (stupeň křivky = 4)	16	16
NURBS křivka (stupeň křivky = 11)	695	780

Rychlost výpočtu a vykreslení všech křivek byla dostačující, kromě křivek NURBS vyšších stupňů. Rychlost křivek NURBS by mohla být optimalizována v některé z budoucích verzí programu s využitím akcelerace pomocí grafické karty, až ji začne podporovat knihovna JOGL. Na výsledcích je zajímavé, že na druhé, méně výkonné sestavě, byly naměřeny vyšší rychlosti (možná rozdíl mezi platformou AMD a Intel).

8 Závěr

Na základě požadavků byl proveden návrh a implementace počítačové aplikace, která umožňuje demonstrovat vlastnosti vybraných křivek a ploch. Návrh i implementace byly provedeny s ohledem na snadnou rozšiřitelnost aplikace o další typy křivek a ploch podle návrhového vzoru MVC.

Funkčnost aplikace byla otestována na operačním systému Microsoft Windows 7 a Ubuntu 10. Při testování bylo odhaleno několik chyb, všechny objevené chyby byly opraveny. Byla otestována i rychlost výpočtů křivek a ploch, naměřené výsledky jsou hodnoceny jako dostačující pro účely výuky.

Implementovaná aplikace umožňuje vybrané křivky a plochy zobrazovat v 3D prostoru, umožňuje libovolně nastavovat pozici kamery, včetně volby mezi ortogonální a perspektivní projekcí. S křivkami a plochami se pracuje formou projektů, každý projekt se může skládat z libovolného počtu křivek a ploch, projekty lze ukládat a načítat do souborů, lze je využít například pro vytvoření ukázkových příkladů. Aby bylo možné vytvořené projekty použít i jiným způsobem, umožňuje aplikace export do obrázku. Aplikace je stabilní, splňuje všechny očekávané vlastnosti a funkce, její výkon je dobrý.

Aplikace by mohla v budoucnu být rozšířena o paletu nástrojů podle vzoru grafických programů, kde by byly nástroje například pro posun objektu, jeho transformace a změnu kamery. Mohla by být přeložena do jiných jazyků, aby ji mohli využívat i zahraniční studenti. Další rozšíření by mohla být implementace celé řady dalších typů křivek a ploch. Během používání aplikace ve výuce, se určitě najde mnoho dalších nápadů a návrhů na zlepšení. Pro usnadnění dalšího vývoje aplikace byly umístěny zdrojové kódy do veřejného repositáře na webu.

9 Seznam použité literatury

[1] Alexandr, Lubomír. *Výuka počítačové grafiky cestou WWW* [online]. Rok neuveden [cit. 2011-04-01]. Dostupné na WWW <http://lubovo.misto.cz/_MAIL_/curves/obsah.html>.

[2] Hoefer, Mark. *Math 199* [online]. Rok neuveden [cit. 2011-04-01]. Dostupné na WWW <http://www.math.ucla.edu/~baker/java/hoefer/>.

[3] Autor neuveden. *Motivating geometry through computation and visualization* [online]. 2002 [cit. 2011-04-01]. Dostupné na WWW <http://www.rose-hulman.edu/~finn/CCLI/applets.htm>.

[4] Žára, Jiří; Beneš, Bedřich; Sochor, Jiří; Felkel, Petr. *Moderní počítačová grafika*. Vydání první. Brno: Computer Press, 2004. ISBN 80-251-0454-0.

[5] Wikipedia. *Polynom* [online]. 2011 [cit. 2011-04-01]. Dostupné na WWW <http://cs.wikipedia.org/wiki/Polynom>.

[6] Wikipedia. *Hermite interpolation* [online]. 2011 [cit. 2011-04-01]. Dostupné na WWW <http://en.wikipedia.org/wiki/Hermite_interpolation>.

[7] Červenka, Milan. *Metoda nejmenších čtverců* [online]. Rok neuveden [cit. 2011-04-01]. Dostupné na WWW <http://herodes.feld.cvut.cz/mereni/mnc/mnc.php>.

[8] Procházková, Jana. *Křivky NURBS* [online]. 2006 [cit. 2011-04-01]. Dostupné na WWW <http://www.root.cz/clanky/krivky-nurbs-1/>.

[9] Altmann, Markus. *About Nonuniform Rational B-Splines* [online]. 1997 [cit. 2011-04-01]. Dostupné na WWW <http://web.cs.wpi.edu/~matt/courses/cs563/talks/nurbs.html>.

[10] Wikipedia. *Non-uniform rational B-spline* [online]. 2011 [cit. 2011-04-01]. Dostupné na WWW <http://en.wikipedia.org/wiki/Non-uniform_rational_B-spline>.

[11] Jáchym, Jakub. *Co je to JAVA?* [online]. 2007 [cit. 2011-04-01]. Dostupné na WWW <http://clanky.katalognotebooku.cz>.

[12] Geeknet, Inc.. *Curve API (CAPI)* [online]. 2005 [cit. 2011-04-01]. Dostupné na WWW <http://curves.sourceforge.net/>.

[13] Université de Montréal. *Stochastic Simulation in Java* [online]. 2010 [cit. 2011-04-01]. Dostupné na WWW <http://www.iro.umontreal.ca/~simardr/ssj/indexe.html>.

[14] CERN - European Organization for Nuclear Research. *Colt* [online]. 2004 [cit. 2011-04-01]. Dostupné na WWW <http://acs.lbl.gov/software/colt/>.

[15] JGEOM. *Geometry library (JGEOM)* [online]. 2010 [cit. 2011-04-01]. Dostupné na WWW <http://www.koders.com/info.aspx?c=ProjectInfo&pid=L7X7GYG8AKNZBSL3H34FWMHGYG>.

[16] JogAmp.org. *JOGL* [online]. 2011 [cit. 2011-04-01]. Dostupné na WWW <http://jogamp.org/jogl/www/>.

[17] Hopkins, Greg. *Java 3D pro začátečníky* [online]. Rok neuveden [cit. 2011-04-01]. Dostupné na WWW <http://www.java3d.org>.

[18] HEJPBézier. *HEJPBézier Library* [online]. 2009 [cit. 2011-04-01]. Dostupné na WWW <http://hejp.co.uk/category/java-library/>.

[19] Autor neuveden. *JAMA : A Java Matrix Package* [online]. 2005 [cit. 2011-04-01]. Dostupné na WWW <http://math.nist.gov/javanumerics/jama/>.

[20] Abeles, Peter. *EJML* [online]. 2011 [cit. 2011-04-01]. Dostupné na WWW <http://code.google.com/p/efficient-java-matrix-library/>.

[21] The Apache Software Foundation. *Batik SVG Toolkit* [online]. 2010 [cit. 2011-04-01]. Dostupné na WWW <http://xmlgraphics.apache.org/batik/>.

[22] CollabNet. *ArgoUML* [online]. 2009 [cit. 2011-04-01]. Dostupné na WWW <http://argouml.tigris.org/>.

[23] Bernard, Borek. *Úvod do architektury MVC* [online]. 2009 [cit. 2011-04-01]. Dostupné na WWW <http://zdrojak.root.cz/clanky/uvod-do-architektury-mvc/>.

[24] BookRags. *Rodrigues' rotation formula* [online]. 2011 [cit. 2011-04-01]. Dostupné na WWW <http://www.bookrags.com/wiki/Rodrigues%27_rotation_formula>.

[25] Tišnovský, Pavel. *Seriál Grafická knihovna OpenGL* [online]. 2003 [cit. 2011-04-01]. Dostupné na WWW <http://www.root.cz/serialy/graficka-knihovna-opengl/>.

10 Přílohy

[F] Složka na CD /prilohy/APIDokumentace: API dokumentace aplikace ve formátu HTML.

[G] Složka na CD /prilohy/cscg: NetBeans projekt hlavní aplikace.

[H] Složka na CD /prilohy/cscgMultiPlatform: Spustitelná aplikace pro Windows a Linux.

[I] Složka na CD /prilohy/Start: NetBeans projekt aplikace pro spuštění hlavní aplikace.

[J] Složka na CD /prilohy/MyObjectLib: NetBeans projekt ukázkové knihovny pro rozšíření.

[A] Návod k používání aplikace

[A.1] Požadavky pro běh aplikace

- Operační systém Windows nebo Linux
- Instalované oficiální ovladače grafické karty
- Instalovanou Javu, instalátor můžete stáhnout na adrese http://www.java.com/en/download/index.jsp
- Rozlišení monitoru minimálně 800x600px

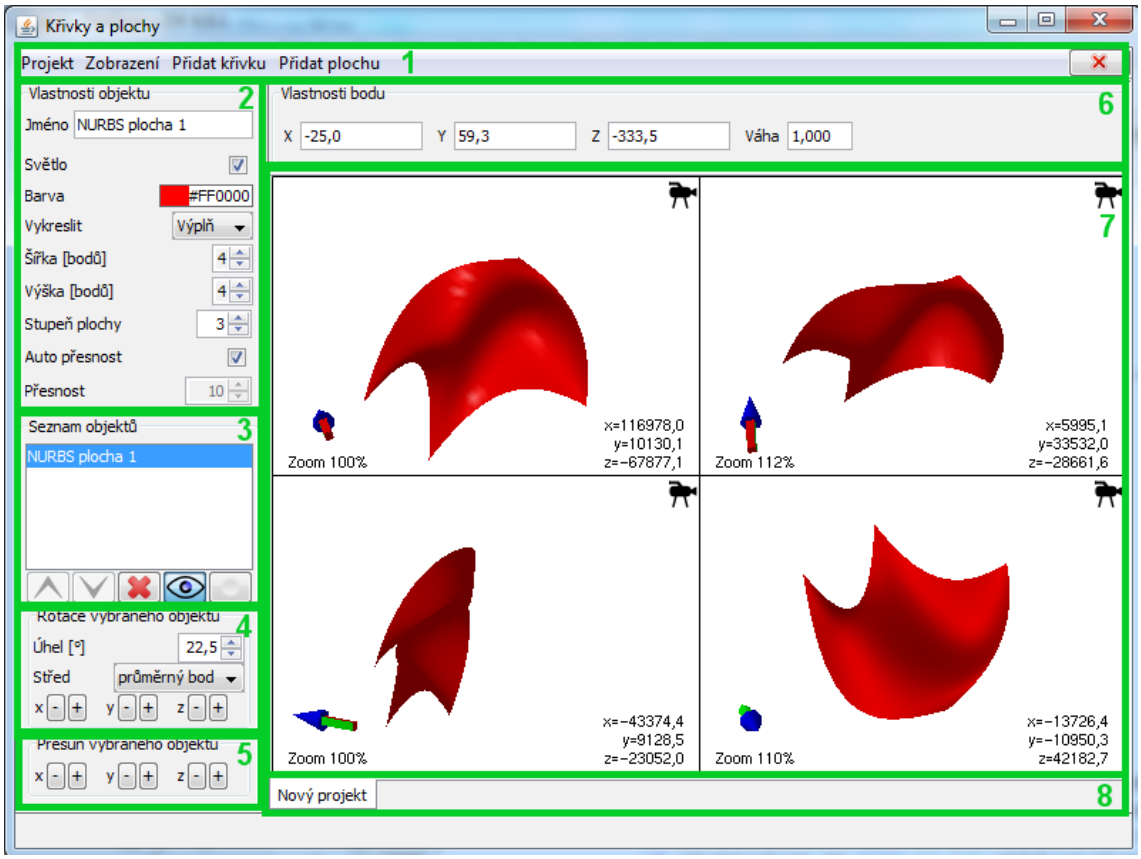
[A.2] Spuštění aplikace

Ve složce programu spusťte soubor Start.jar.

[A.3] Popis oken aplikace

Hlavní okno aplikace

1. Hlavní menu – struktura menu:
    - Projekt
        - Nový projekt – vytvoří nový projekt.
        - Otevřít projekt – otevře projekt uložený v počítači.
        - Uložit projekt – uložení aktuálně vybraného projektu.
        - Uložit jako – uloží aktuálně vybraný projekt do nového souboru.
        - Exportovat obrázek – uloží obrázek aktuálně vybraného projektu ve formátu PNG.
        - Zavřít projekt – zavře aktuálně vybraný projekt.
        - Konec – ukončí aplikaci. Aplikace si při ukončení ukládá aktuální stav a při příštím spuštění se bude nacházet ve stejném stavu.
    - Zobrazení – všechny volby v tomto podmenu ovlivňují pouze aktuálně vybraný projekt.
        - Jeden pohled / Kombinace – volba mezi rozdělením editoru na čtyři pohledy nebo jeden.
        - Režim editace – přepnutí editoru do režimu editace. V režimu editace se v editoru vykreslují řídící body.
        - Vykreslit osy – přepnutí vykreslení osového kříže v editoru.
        - Vykreslit informační text – přepnutí zobrazení textu informujícího o aktuálně nastaveném zoomu a pozici kurzoru v editoru.
        - Vykreslit orientační ikonu – přepnutí zobrazení orientační ikony v editoru, která znázorňuje orientaci kamery.
        - Otevřít seznam bodů – otevře okno pro editaci řídících bodů a uzlových vektorů.
        - Nastavení – otevře okno pro nastavení programu.

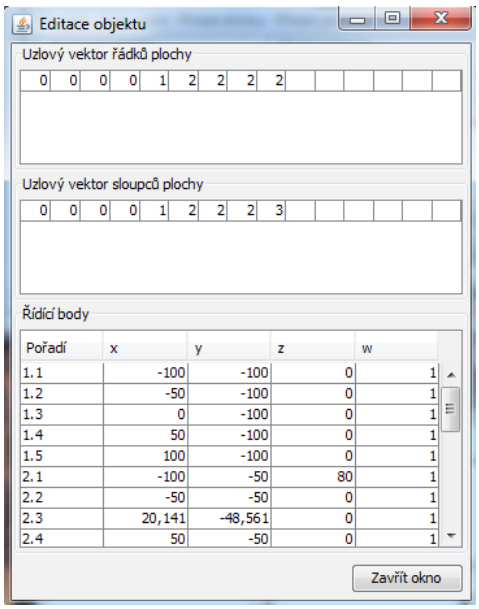


Obrázek 10.1: Hlavní okno aplikace

- Podmenu pro přidání objektu do projektu. Liší se podle konfigurace aplikace.
- Tlačítko pro zavření aktuálně zobrazeného projektu.

2. Vlastnosti vybraného objektu – liší se podle aktuálně vybraného objektu.
3. Seznam vytvořených objektů v projektu – v tomto panelu vyberte objekt, který chcete editovat. Popis tlačítek:
   - Přesun objektu nahoru – přesune vybraný objekt v seznamu výše, objekty, které jsou v seznamu výše budou vykreslený v nejvyšší vrstvě.
   - Přesun objektu dolů – přesune vybraný objekt v seznamu směrem dolů.
   - Smazání objektu – smaže vybraný objekt.
   - Viditelnost objektu – přepne vykreslování vybraného objektu.
   - Exkluzivní viditelnost – schová všechny objekty v projektu a vykreslí pouze aktuálně vybraný objekt. Nastavení platí, dokud nedojde ke změně vybraného objektu.
4. Rotace vybraného objektu – rotuje aktuálně vybraný objekt v prostoru. Můžete nastavit úhel a střed rotace: průměrný bod je bod objektu vypočtený jako aritmetický průměr všech řídících bodů objektu, střed objektu je bod vypočtený jako bod uprostřed prostoru vymezeného nejvzdálenějšími řídícími body objektu, počátek je bod se souřadnicemi [0;0;0]. Tlačítko plus a mínus udávají směr rotace a jsou rozděleny do skupin podle osy rotace.

5. Přesun vybraného objektu – přesune všechny řídící body vybraného objektu daným směrem.
6. Vlastnosti bodu – panel umožňuje nastavit pozici vybraného řídícího bodu a další vlastnosti, pokud je objekt podporuje, například váhu bodu.
7. Editor – vykresluje aktuálně vybraný projekt.
8. Záložky projektů – každý otevřený projekt má svojí záložku.



Obrázek 10.2: Okno pro nastavení řídících bodů a uzlových vektorů

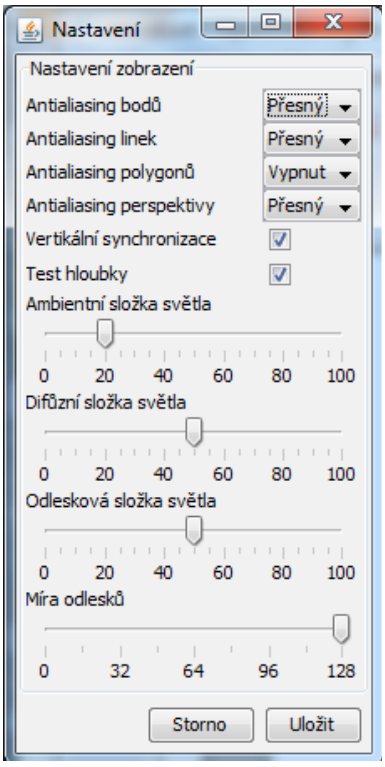
Okno pro nastavení řídících bodů a uzlových vektorů

Okno můžete zobrazit přes hlavní menu → Zobrazení → Otevřít seznam bodů. V okně se vždy zobrazí řídící body a uzlové vektory aktuálně vybraného objektu. Řídící body i uzlové vektory můžete v okně přímo měnit.

Popis okna:

- Řídící body – tabulka řídících bodů objektu. Pokud je objekt křivka, je v prvním sloupci zobrazeno pořadí bodu, pokud je objekt plocha, jsou v prvním sloupci uvedeny souřadnice do mřížky řídících bodů, kde první číslo je číslo řádku a číslo za tečkou je číslo sloupce. V následujících sloupcích jsou zobrazeny souřadnice *x, y* a *z,* pokud mají řídící body i váhu, pak je zobrazena v posledním sloupci. Pokud kliknete na některý řádek s řídícím bodem, daný bod se zvýrazní i v okně editoru, vazba funguje i obráceně.

- Uzlové vektory – tabulka zobrazující uzlové vektory. Pro křivky je zobrazen jeden uzlový vektor, pro plochy dva. Uzlové vektory jsou zobrazeny pouze pokud je objekt podporuje.



Obrázek 10.3: Okno pro nastavení aplikace

Okno pro nastavení aplikace

Okno můžete zobrazit přes hlavní menu → Zobrazení → Nastavení. V okně nastavení můžete nastavit kvalitu vykreslování a vlastnosti světla. Nastavení vlastností světla je velice užitečně při vykreslování ploch. Vlastnosti světla

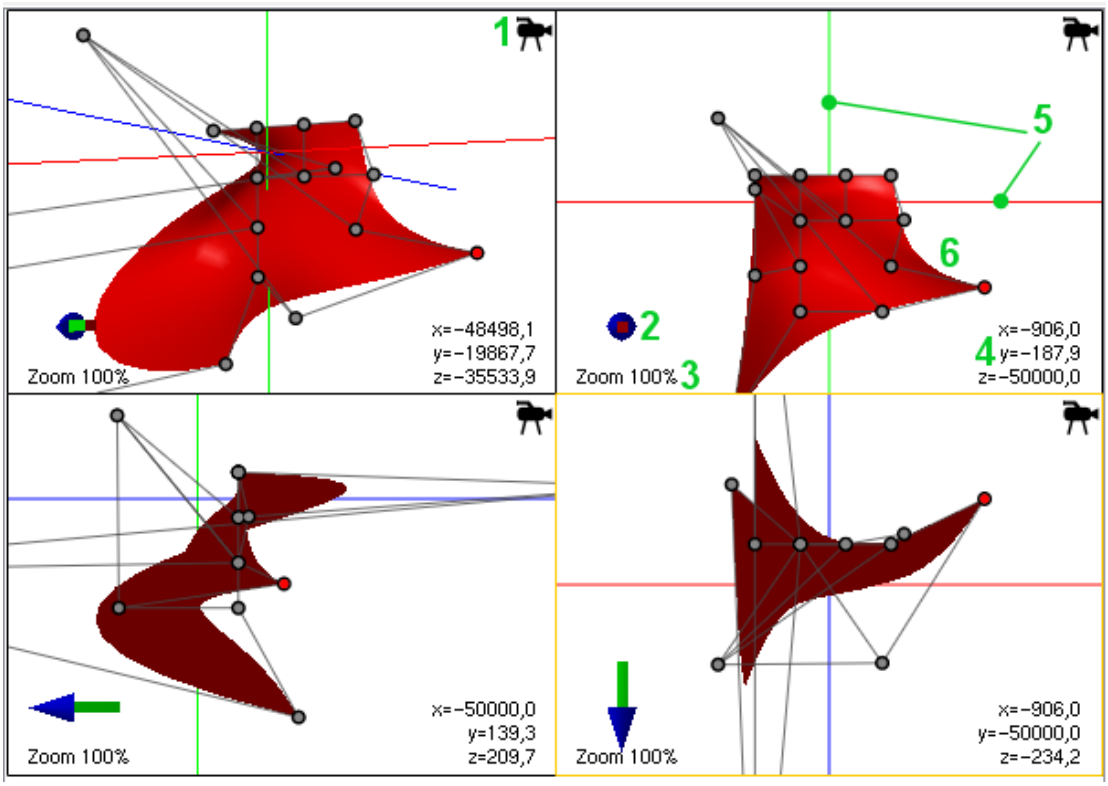
se nastavují podle Phongova osvětlovacího modelu. Změny nastavení se projeví ihned, tlačítkem Uložit změny budou potvrzeny a tlačítkem Storno budou změny zrušeny.

Editor

Editor je součástí hlavního okna. Vykresluji vždy aktuální projekt. Popis částí editoru:

1. Ikona kamery – kliknutím na ikonu otevřete menu kamery, ve kterém můžete přepnout mezi perspektivní a ortogonální projekcí a nastavit směr pohledu kamery.
2. Orientační ikona kamery – šipka znázorňující aktuální natočení kamery v prostoru.
3. Informační text zobrazující aktuální zoom.
4. Informační text ukazující souřadnice na kterých se nachází kurzor myši.
5. Osový kříž – osa *x* je červená, *y* je zelená a *z* je modrá.
6. Vykreslený objekt a řídící body. Aktuálně vybraný řídící bod je zvýrazněn červeným středem.

Jednotlivé pohledy editoru jsou odděleny rámečkem. Pohled který má aktuálně fokus má rámeček oranžové barvy.



Obrázek 10.4: Editor se čtyřmi pohledy

[A.4] Ovládání editoru

Myš

- Pravé tlačítko + táhnutí – přesun pozice kamery.
- Pravé tlačítko + CTRL + táhnutí nebo prostřední tlačítko + táhnutí – rotace kamery kolem aktuálně vybraného objektu.
- Levé tlačítko – výběr řídících bodů a jejich přesun
  - Levé tlačítko – výběr jednoho řídícího bodu
  - CTRL + levé tlačítko – přidání řídícího bodu do výběru
  - SHIFT + levé tlačítko – nepřidá řídící bod do výběru
  - SHIFT + CTRL + levé tlačítko – zablokuje výběr řídících bodů proti změně
- Kolečko – při ortogonální projekci změní zoom, při perspektivní projekci posune pozici kamery.

Klávesnice

- F1 – přepnutí mezi perspektivní a ortogonální projekcí
- F2 – nastaví pohled kamery zepředu
- F3 – nastaví pohled kamery zleva
- F4 – nastaví pohled kamery shora
- Šipky – přesun aktuálně vybraných řídících bodů
- 2, 8 – přesun pozice kamery dolů a nahoru
- 4, 6 – přesun pozice kamery doleva a doprava
- 1, 7 – přesun pozice kamery dopředu a dozadu
- 3, 9 – změna natočení kamery nahoru a dolů
- 0, čárka, tečka – změna natočení kamery doleva a doprava
- /, * - rotace kamery kolem své osy
- +, mínus – změna zoomu
- 5 – resetování nastavení kamery

[B] Postup pro rozšíření aplikace o nový typ objektu

Rozšířit aplikaci o další typy křivek a ploch můžeme dvěma způsoby: buď upravíme aplikaci samotnou nebo vytvoříme novu knihovnu, kterou k aplikaci připojíme.

Popis souvisejících příloh na CD:

- Příloha [F]: API dokumentace.
- Příloha [G]: projekt aplikace.
- Příloha [H]: zkompilovaná spustitelná aplikace.
- Příloha [J]: projekt ukázkové knihovny pro rozšíření aplikace.

[B.1] Rozšíření úpravou aplikace

1. V této podkapitole je popsán postup, při kterém upravíme samotný projekt aplikace.
2. Zkopírujte si na pevný disk z příloh složky cscg a cscgMultiPlatform.
3. V NetBeans otevřete projekt z kopie složky cscg.
4. Přidejte do balíčku **cscg.model.objects.impl** novou třídu, která bude představovat novou křivku nebo plochu. Popis jak implementovat samotnou třídu je popsán v kapitole B.3. Tento krok je stejný jako při rozšíření aplikace externí knihovnou.
5. Zkompilujte projekt.
6. Zkompilovaný soubor cscg.jar (NetBeans vytváří jar soubor do složky dist ve složce projektu) zkopírujte do složky cscgMultiPlatform, kterou jste si z CD zkopírovali v prvním kroku.
7. Přidáme novou křivku do menu aplikace. Postup je vysvětlený v kapitole B.4.

[B.2] Příprava projektu pro vytvoření rozšiřující knihovny

1. V této podkapitole je popsán postup, rozšíření aplikace pomocí externí knihovny.
2. Zkopírujte si na pevný disk z příloh složky MyObjectLib a cscgMultiPlatform.
3. V NetBeans otevřete projekt MyObjectLib, je to připravený ukázkový projekt pro rozšíření aplikace.
4. Do balíčku **mylib** vytvořte novou třídu, která bude představovat novou křivku nebo plochu. Popis jak implementovat samotnou třídu je popsán v kapitole B.3.
5. Zkompilujte projekt.
6. Zkompilovaný soubor MyObjectLib.jar (NetBeans vytváří jar soubor do složky dist ve složce projektu) zkopírujte do složky cscgMultiPlatform/lib.
7. Otevřete soubor cscgMultiPlatform/cscg.jar jako zip archív, například pomocí programu 7-zip, který lze zdarma stáhnout na adrese http://www.7-zip.org/. V archívu upravte soubor /META-INF/MANIFEST.MF tak že k hodnotě proměnné **Class-Path** přidáte naší novou knihovnu, takže přidejte například lib/MyObjectLib.jar.
8. Přidáme novou křivku do menu aplikace. Postup je vysvětlený v kapitole B.4.

[B.3] Implementace nové křivky nebo plochy

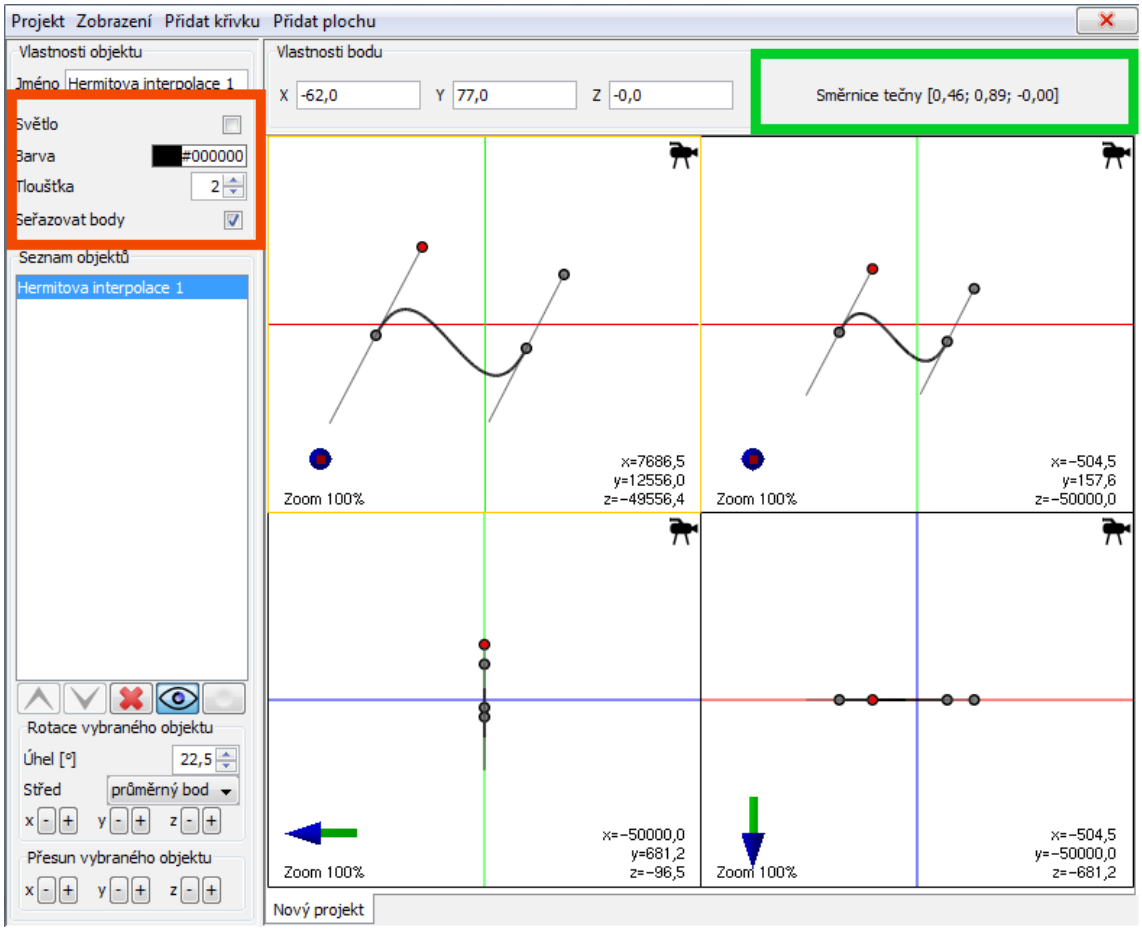
Každý objekt projektu musí implementovat rozhraní **IObject**. Aplikace je vícevláknová, s objekty projektu pracují minimálně dvě vlákna (vlákno starající se o vykreslení a vlákno zpracovávající vstupní události), proto tomu musíme přizpůsobit metody. Nejjednodušší způsob je deklarovat metody jako synchronized. Každý implementovaný objekt se skládá z několika částí:

1. GUI pro nastavení vlastností objektu (například stupeň křivky, barva a další vlastnosti, které se mohou lišit pro každý typ objektu). Objekt nemusí mít žádné GUI. Na obrázku 10.5 je ukázka GUI vlastností objektu zvýrazněna červeně.
2. GUI pro nastavení vlastností bodu, slouží pro nastavení vlastností jiných než souřadnice nebo váha bodu (GUI pro nastavení souřadnic a váhy bodů vytváří automaticky hlavní okno aplikace), většina objektů toto GUI vůbec nebude využívat, ale například objekt Hermitovy interpolace v tomto GUI zobrazuje směrnici tečny v daném bodě. Na obrázku 10.5 je ukázka GUI bodu zvýrazněna zeleně.
3. Seznam řídících bodů, všechny body musí implementovat rozhraní IPoint3f.
4. Seznam řídících bodů, které jsou ve výběru (výběrem je myšlen výběr bodů uživatelem, například kliknutím na bod v editoru).

Všechny objekty musí být serializovatelné, protože rozhraní IObject rozšiřuje rozhraní **java.io.Serializable**, serializace objektů se využívá pro ukládání projektů a stavu aplikace. Kvůli serializaci musíme zajistit aby všechny proměnné objektů byly serializovatelné nebo transientní.

Pro zjednodušení vytváření nových křivek a ploch, jsou vytvořeny tři abstraktní třídy. Abstraktní třída **AbstractObject** je základní implementace rozhraní IObject. Implementuje všechny vlastnosti objektů, které jsou společné pro křivky i plochy: barvu, osvětlení, metody pro posun bodů, pro nastavení výběru bodů, rotaci a posun objektu. Třída je generická a může pracovat s libovolnými body, které dědí z třídy **Point3f**, toho využívá například implementace NURBS křivky a plochy, která používá body **Point4f**.

Pokud budeme mít potomka třídy, který bude používat body jiné třídy než Point3f, musíme do chráněné instanční proměnné **pointPrototype** uložit instanci jednoho bodu. Třída vytváří nové body do objektu jako klony tohoto bodu, tím je zajištěno že objekt může využívat libovolné body a zároveň můžeme přednastavit základní vlastnosti bodu, například pro NURBS křivky a plochy je vytvořený **pointPrototype** jako instance bodu Point4f s nastavenou vahou 1, takže nemusíme zvlášť pro každý nový bod nastavovat základní váhu.



Obrázek 10.5: Aplikace se zvýrazněním prostoru pro GUI vlastností objektu a bodu

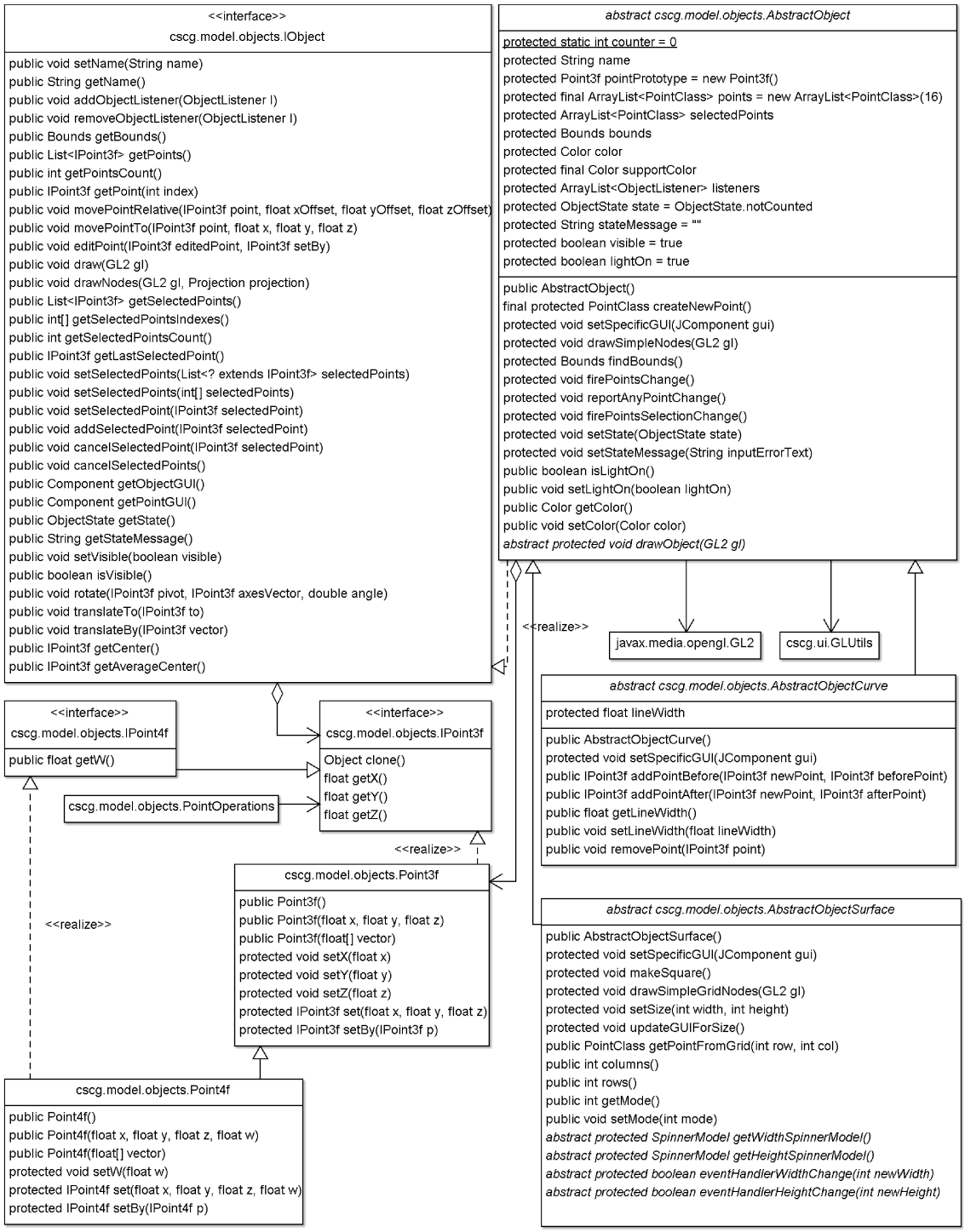
Další velice užitečnou třídou je abstraktní třída **AbstractObjectCurve**. Tato třída rozšiřuje AbstractObject o vlastnosti společné všem křivkám: tloušťka čáry, přidání a odebrání řídícího bodu. Pokud budeme implementovat novou křivku, je rozšíření této třídy nejlepší a nejrychlejší volba.

Poslední abstraktní třídou je AbstractObjectSurface. Tato třída rozšiřuje AbstractObject o vlastnosti společné pro plochy: způsob vykreslení plochy – síť, body, výplň, implementuje rozhraní ISurface, které obsahuje metody pro zjištění počtu sloupců a řádků řídících bodů plochy, metody pro získání řídícího bodu podle souřadnic v rámci sítě řídících bodů. Tuto třídu budeme rozšiřovat pokud budeme do aplikace přidávat novou plochu.

Následující třídy souvisí s implementací nových křivek a ploch:

- cscg.model.object.PointOperations – implementuje výpočty s body a vektory.
- cscg.model.object.MathUtils – implementuje některé obecné funkce jako faktoriál, kombinace.

- cscg.ui.GLUtils – implementuje operace usnadňující práci s OpenGL.



Obrázek 10.6: Diagram tříd souvisejících s rozšířením aplikace

- Podrobný seznam a popis metod můžete nalézt v API dokumentaci.

[B.3.1] Příklad 1: nejjednodušší implementace křivky

Tento příklad se týká ukázkové třídy **mylib.ObjectLine** z ukázkové knihovny. Výsledkem je nejjednodušší implementace křivky jaké lze dosáhnout.

Vytváříme křivku, proto budeme dědit z třídy cscg.model.objects.AbstractObjectCurve. Křivka bude tvořena obyčejnými body Point3f.

```
public class ObjectLine extends AbstractObjectCurve<Point3f>{}
```

Každá nově vytvořená instance bude pojmenovaná podle pořadí vytvoření, proto definujeme statické počítadlo.

```
private static int selfCounter=1;
```

V konstruktoru nastavíme základní vlastnosti křivky – tloušťku čáry, inicializujeme stav křivky a jméno křivky.

```
super();
setName("Úsečky "+selfCounter++);
setState(ObjectState.OK);
```

Stav křivky informuje, zdali je křivka zadaná v pořádku, pokud například uživatel zadá takové řídící body, že křivka neexistuje, pak stav nastavíme na hodnotu ObjectState.inputError.

Nyní přidáme metodu, která je volána jednorázově při změně objektu, v těle metody musí dojít k výpočtu křivky a k jejímu vykreslení. V metodě dojde ke kontrole jestli je dostatek řídících bodů, pokud ano, tak se nastaví tloušťka čáry a barva a poté se v cyklu postupně vykreslí čára pomocí OpenGL. Tělo této metody je správné místo pro implementaci algoritmu křivky.

```
@Override
public synchronized void drawObject(GL2 gl)
{
	if(points.size()>1)//musí být zadány alespoň 2 body
	{
		gl.glLineWidth(lineWidth);
		gl.glBegin(gl.GL_LINE_STRIP);
		GLUtils.glSetColor(gl, color);
		for(IPoint3f p:points)
		{
			gl.glVertex3f(p.getX(),p.getY(),p.getZ());
		}
		gl.glEnd();
	}
}
```

Nyní přidáme metodu pro vykreslení editačního rozhraní – vykreslení řídících bodů a dalších prvků, například řídícího polygonu atd. Tato metoda je volána jednou pro každý pohled vždy po změně objektu nebo při změně pohledu. Jejím vstupním parametrem je aktuální projekce. Ve většině případů nepotřebujeme projekci vůbec znát, ale například pro vykreslení šipek v 3D prostoru bude potřeba. V metodě dojde pouze k zavolání funkce drawSimpleNodes z třídy AbstractObject, tato metoda vykreslí všechny řídící body.

```
@Override
public synchronized void drawNodes(GL2 gl, Projection projection)
{
	drawSimpleNodes(gl);
}
```

Tím máme implementován kompletní objekt, který bude v aplikaci fungovat.

[B.3.2] Příklad 2: použití bodů s vahou a omezení pozice bodů

Tento příklad se týká ukázkové třídy **mylib.ObjectLine2** z ukázkové knihovny. Tato třída je rozšíření předcházejícího příkladu o použití jiné třídy bodů než Point3f a omezením pozice.

Křivka bude určena řídícími body třídy MyPoint4f.

```
public class ObjectLine2 extends AbstractObjectCurve<MyPoint4f> {}
```

Do konstruktoru musíme přidat nastavení vzoru nových bodů. Toto nastavení musíme udělat pro každý objekt, který používá body jiného typu než Point3f. Třída AbstractObject vytváří nové řídící body klonováním vzoru. Zároveň vzoru můžeme nastavit základní vlastnosti bodu, v tomto případě například váhu bodu.

```
MyPoint4f p=new MyPoint4f();
p.setW(0.5f);
pointPrototype=p;
```

Nyní přidáme funkci pro kontrolu polohy bodu. Omezíme polohu bodu na souřadnice *x* v rozmezí od 0 do 200. Funkce bude vracet nový bod s upravenými souřadnicemi.

```
private IPoint3f checkPosition(IPoint3f point)
{
	MyPoint4f newPoint=createNewPoint();//nový bod podle vzoru
	newPoint.setBy(point);
	if(newPoint.getX()<0)
	{
		newPoint.setX(0);
	}
	if(newPoint.getX()>200)
	{
		newPoint.setX(200);
	}
	return newPoint;
}
```

Nyní musíme přepsat všechny metody, které mohou měnit řídící body, celkem jich je pět. Do těla každé metody přidáme kontrolu pozice bodu.

```
@Override
public synchronized IPoint3f addPointAfter(IPoint3f newPoint, IPoint3f
afterPoint)
{
	newPoint=checkPosition(newPoint);//kontrola polohy
	return super.addPointAfter(newPoint, afterPoint);
}

@Override
public synchronized IPoint3f addPointBefore(IPoint3f newPoint, IPoint3f
beforePoint)
{
	newPoint=checkPosition(newPoint);//kontrola polohy
	return super.addPointBefore(newPoint, beforePoint);
}

@Override
public synchronized void editPoint(IPoint3f editedPoint, IPoint3f setBy)
{
	setBy=checkPosition(setBy);//kontrola polohy
	super.editPoint(editedPoint, setBy);
}

@Override
public synchronized void movePointRelative(IPoint3f point, float xOffset,
float yOffset, float zOffset)
{
	super.movePointRelative(point, xOffset, yOffset, zOffset);
	editPoint(point, point);//kontrola polohy
}

@Override
public synchronized void movePointTo(IPoint3f point, float x, float y, float
z)
{
	super.movePointTo(point, x, y, z);
	editPoint(point, point);//kontrola polohy
}
```

Tím je celá úprava křivky hotová. Protože jsme použili body třídy MyPoint4f, která implementuje rozhraní IPoint4f, bude se v GUI pro nastavení bodu zobrazovat kolonka pro nastavení váhy bodu.

[B.3.3] Příklad 3: nejjednodušší implementace plochy

Implementace plochy je ve většině ohledů stejná jako implementace křivky, proto popíšu pouze rozdíly. Příklad se týká ukázkové třídy **mylib.ObjectSimpleSurface** z ukázkové knihovny.

Při implementaci plochy rozšíříme třídu **cscg.model.objects.AbstractObjectSurface**. Plocha bude dána řídícími body třídy Point3f.

```
public class ObjectSimpleSurface extends AbstractObjectSurface<Point3f>{}
```

V konstruktoru vytvoříme základní tvar plochy – čtverec, pomocí funkce makeSquare. Dále musíme implementovat metodu vracející spinner model použitý pro GUI prvek pro nastavení počtu sloupců a řádků plochy. Zde vytvoříme spinner model, který bude nastavitelný od hodnoty 2 do 100 krokem 1 a základní hodnota bude 2.

```
@Override
protected synchronized SpinnerModel getWidthSpinnerModel()
{
	return new EnumNumberSpinnerModel(2, 2, 1, 100);
}

@Override
protected synchronized SpinnerModel getHeightSpinnerModel()
{
	return new EnumNumberSpinnerModel(2, 2, 1, 100);
}
```

Dále musíme implementovat metody, které jsou vyvolány při změně hodnoty GUI prvku pro nastavení počtu sloupců nebo řádků. Metody volá automaticky předek **AbstractObjectSurface**. V těle metody zavoláme metodu setSize, která změní počet řídících bodů plochy. Vrátíme hodnotu true, která znamená potvrzení změny počtu řídících bodů.

```
@Override
protected synchronized boolean eventHandlerWidthChange(int newWidth)
{
	setSize(newWidth,rows());
	return true;
}

@Override
protected synchronized boolean eventHandlerHeightChange(int newHeight)
{
	setSize(columns(),newHeight);
	return true;
}
```

Nakonec ještě musíme implementovat metody pro samotné vykreslení plochy a řídících bodů. Metoda pro vykreslení plochy využívá metodu GLUtils.drawSurface, která vykreslí plochu zadanou sítí bodů (více v API dokumentaci). Metoda pro vykreslení řídících bodů volá metodu drawSimpleGridNodes, kterou implementuje předek **AbstractObjectSurface**, tato metoda vykreslí řídící síť.

```
@Override
public synchronized void drawObject(GL2 gl)
{
	GLUtils.glSetColor(gl, color);
	GLUtils.drawSurface(gl, points, columns(), rows(), getMode());
```

```
}

@Override
public synchronized void drawNodes(GL2 gl, Projection projection)
{
	drawSimpleGridNodes(gl);
}
```

Nyní máme implementovánu nejjednodušší plochu, kterou můžeme do aplikace přidat.

[B.3.4] Příklad 4: přidání GUI vlastností objektu a bodu

V předchozích příkladech bylo vysvětleno jak implementovat křivky a plochy. Poslední věc, která v příkladech chybí, je vlastní GUI. GUI využijeme například u objektů, pro které potřebujeme nastavit stupeň křivky nebo libovolné jiné vlastnosti.

Při každé změně objektu, kdy se změní nějaká vlastnost, kterou jsme objektu přidali, musíme oznámit změnu objektu všem posluchačům voláním metody všech posluchačů ObjectListener.eventSpecificPropertiesChanged(this).

Tento příklad se týká ukázkové třídy **mylib.ObjectLineWithGUI** z ukázkové knihovny. Tato třída je rozšíření prvního příkladu.

Objekt rozšíříme o možnost nastavit stupeň křivky. Stupeň křivky se bude nastavovat v GUI pro nastavení vlastností objektu. Dále přidáme GUI pro nastavení vlastností bodu, ve kterém zobrazíme pořadí aktuálně vybraného bodu. Deklaruje si třídní proměnné pro uložení stupně křivky a GUI prvky. Protože GUI prvky se nebudou serializovat, vytvoříme si metodu initTransients, ve které inicializujeme všechny transientní vlastnosti. Tuto metodu budeme volat v konstruktoru a při deserializaci.

```
/**
 * GUI pro nastavení stupně křivky.
 */
private transient JSpinner guiDegree;
/**
 * Stupeň křivky.
 */
private volatile int degree = 3;
/**
 * GUI panel pro vlastnosti bodu
 */
private transient JPanel pointPanel;
/**
 * GUI label zobrazující pořadí bodu
 */
private transient JLabel pointLabel;
/**
 * Konstruktor
 */
public ObjectLineWithGUI()
{
```

```
        super();
        setName("Úsečky s GUI "+selfCounter++);
        initTransients();
}
/**
 * Inicializace transientních vlastností
 */
private void initTransients()
{
        initGUI();
}
/**
 * Deserializace
 */
private void readObject(ObjectInputStream in) throws IOException,
ClassNotFoundException
{
        in.defaultReadObject();
        initTransients();
}
```

V těle metody initTransients voláme metodu initGUI, která vytváří samotné GUI. Tělo celé této metody zde nebude uvedeno, jen vyzdvihnu důležité části.

```
private void initGUI()
{
        //panel s nastavením vlastností objektu
        JPanel guiSetings=new JPanel(new GridBagLayout());

        //... vytváření GUI prvků přeskočeno

        //panel s nastavením vložím do nadřazeného gui
        super.setSpecificGUI(guiSetings);

        //posluchač změn GUI pro nastavení stupně
        guiDegree.addChangeListener(new ChangeListener()
        {
                @Override
                public void stateChanged(ChangeEvent e)
                {
                        setDegree((Integer) guiDegree.getValue());
                }
        });

        //inicializace gui pro vlastnosti bodu
        pointPanel=new JPanel(new FlowLayout(FlowLayout.LEFT, 0, 0));
        pointLabel=new JLabel();
        pointPanel.add(pointLabel);
}
```

Nyní přidáme metodu, která vrací GUI pro nastavení vlastností bodu.

```
@Override
public Component getPointGUI()
{
        return pointPanel;
}
```

Protože rozšiřujeme křivku o nastavení stupně křivky, musíme přidat metody pro nastavení stupně voláním metodou.

```
/**
 * Získání stupně křivky
 */
public synchronized int getDegree()
{
	return degree;
}

/**
 * Nastavení stupně křivky.
 */
public synchronized void setDegree(int degree)
{
	this.degree = degree;
	guiDegree.setValue(degree);
	setState(ObjectState.notCounted);//křivka se změnila
	//oznámení změny
	for (ObjectListener l : listeners)
	{
		l.eventSpecificPropertiesChanged(this);
	}
}
```

Poslední úprava je implementace metody, která je volána pokaždé, když dojde ke změně výběru bodů nebo vlastností bodu. Tuto metodu využijeme k aktualizaci GUI pro nastavení vlastností bodu.

```
/**
 * Sledování změn bodu
 */
@Override
protected synchronized void reportAnyPointChange()
{
	int selectedSize=selectedPoints.size();
	//mám vybráno mnoho bodů nebo žádný
	if(selectedSize!=1)
	{
		pointLabel.setText("Vyberte jeden bod");
		return;
	}
	//mam vybrán 1 bod
	//index bodu
	int index=points.indexOf(selectedPoints.get(0));
	pointLabel.setText("Pořadí bodu je: "+index);
}
```

Tím jsme dokončili rozšíření objektu o vlastní GUI a můžeme křivku přidat do projektu.

[B.3.5] Další zdroje příkladů

Jako další příklady můžete využít konkrétní implementace křivek a ploch, které se nachází v aplikaci v balíčku **cscg.model.objects.impl**.

[B.4] Přidání křivky nebo plochy do menu aplikace

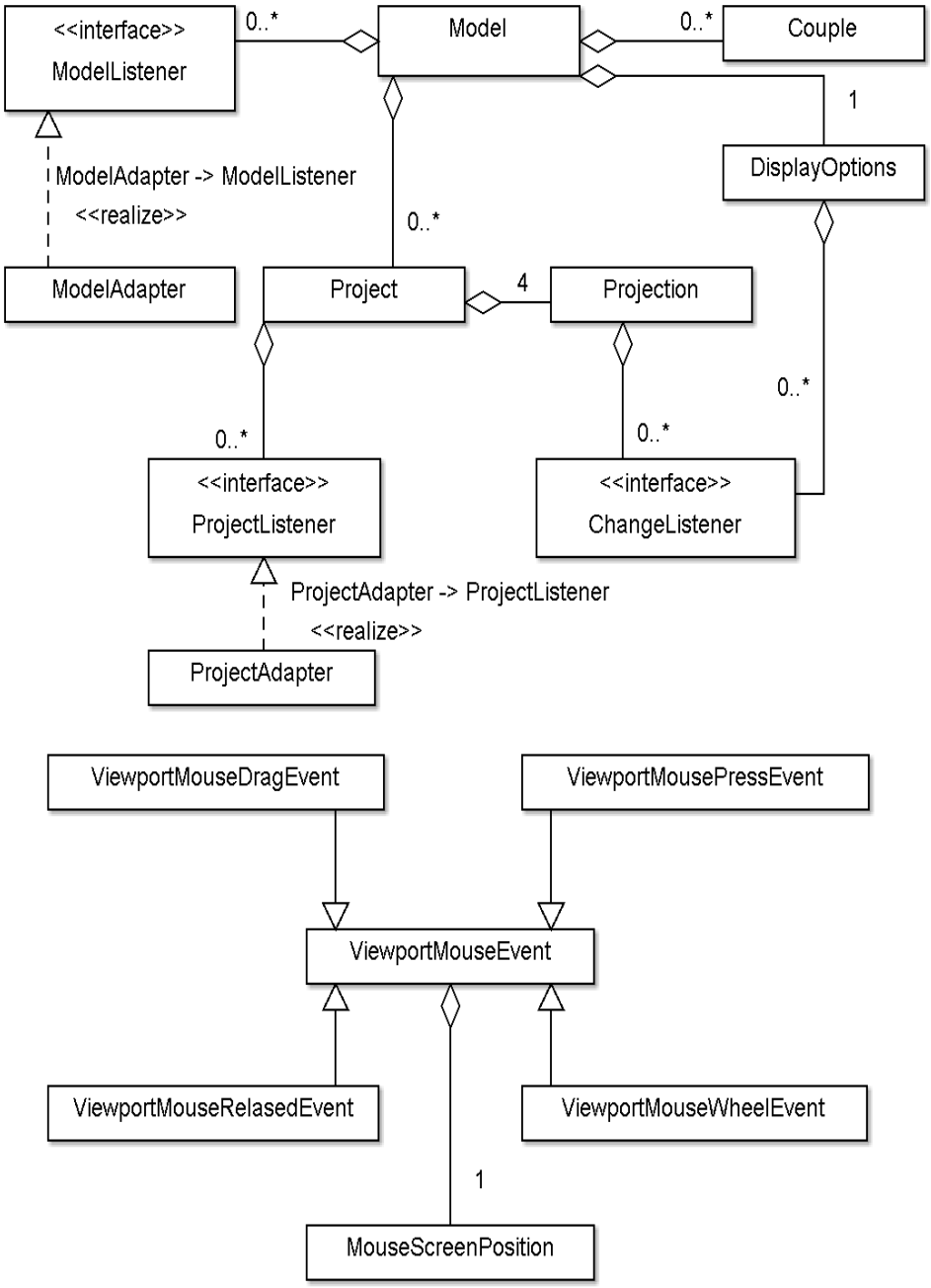
V předchozích kapitolách bylo vysvětleno jak implementovat nové křivky a plochy a přidat je do aplikace. Abychom mohli tyto křivky využívat v projektech, chybí nám udělat poslední úprava: přidat do menu aplikace položku pro přidání našeho nového objektu. Konfigurace objektů v menu aplikace je uložena v XML souboru.

1. Otevřete soubor cscgMultiPlatform/cscg.jar jako zip archív, například pomocí programu 7-zip. V archívu upravte soubor /menu.xml. Soubor má formát podle DTD difinice v souboru /menu.dtd.
2. Soubor /menu.xml má kořenový element <menu>. Nové podmenu vytvoříte elementem <submenu title="Moje knihovna">, kde atribut title obsahuje text, který se zobrazí jako název podmenu. Uzel <submenu> může obsahovat libovolný počet objektů nebo separátorů. Separátor má značku <separator /> a v menu vytvoří oddělovač. Objekt má značku <object class="mylib.ObjectLine" title="Můj objekt" />, atribut class udává třídu objektu a atribut title udává text, který se zobrazí v menu. Soubor \menu.xml, který by definoval menu objektů naší ukázkové knihovny může vypadat následovně:

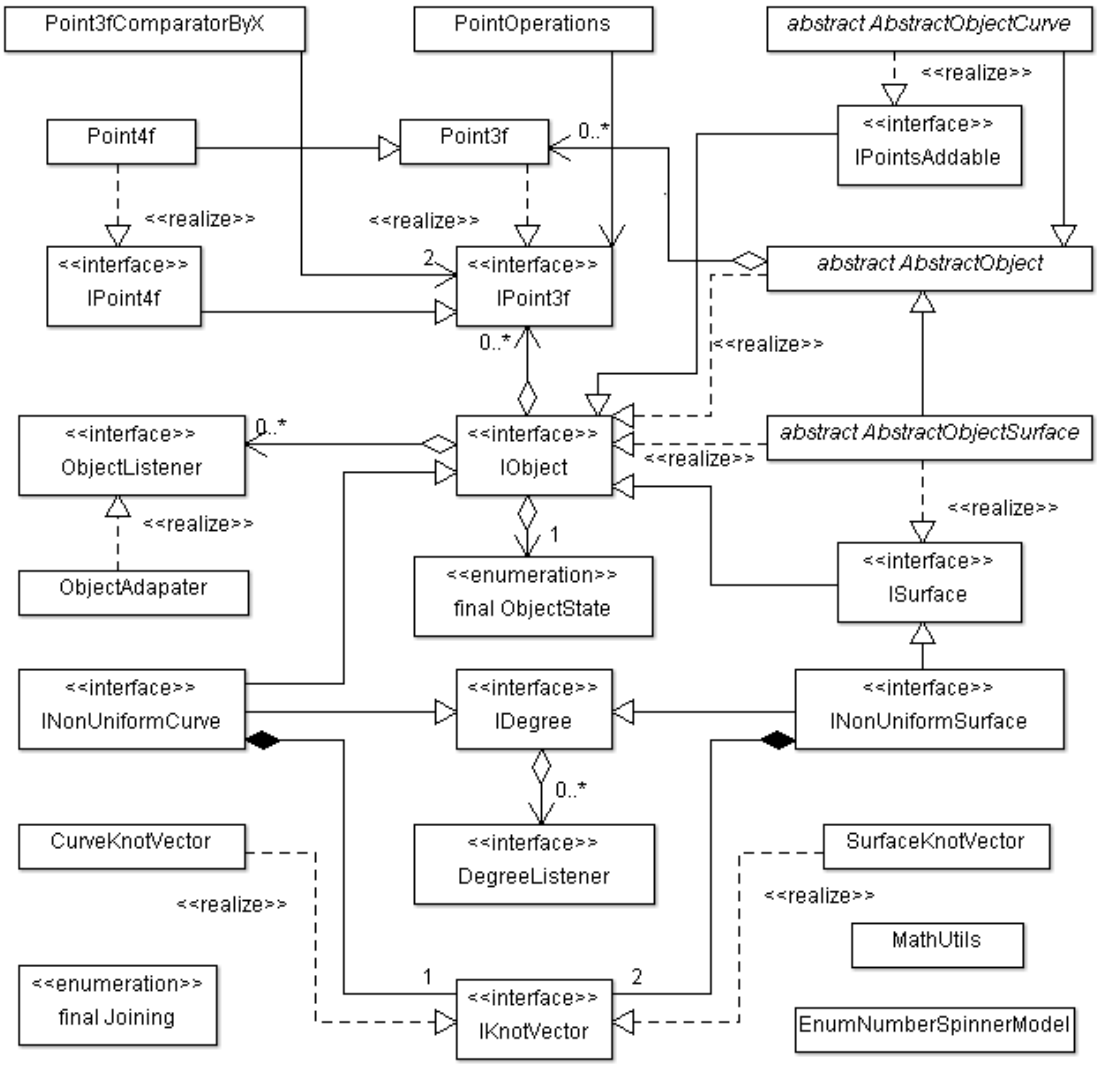
```
<menu>
	<submenu title="Moje knihovna">
		<object class="mylib.ObjectLine" title="Příklad 1" />
		<object class="mylib.ObjectLine2" title="Příklad 2" />
		<object class="mylib.ObjectLineWithGUI" title="Příklad 4" />
		<object class="mylib.ObjectSimpleSurface" title="Příklad 3" />
	</submenu>
</menu>
```

Tím je přidání nového objektu dokončeno a můžete ho využívat.

[C] Diagram tříd balíčku cscg.model



[D] Diagram tříd balíčku cscg.model.objects



[E] Diagram tříd balíčku cscg.model.objects.impl